

Chapter Five: Contents**(AS-3 – 10 December 2002 – LA-UR 01-5716 – Portland Study Reports)**

1. CONFIGURATION FILES	1
1.1 ALLSTR_ACTGEN_RIVER_CROSS.CFG	1
2. SCRIPTS	6
2.1 SPLITACTBYZONE.PL	6
2.2 FINDCROSSINGS.PL	8
2.3 FIXBRIDGECROSS.NS	13
2.4 FIXBRIDGECROSS.SN	19
2.5 FIXBRIDGECROSS.WE	25
2.6 FIXBRIDGECROSS.EW	29
3. TRAVEL TIME FUNCTIONS	33
3.1 TRAVELTIMEFUNCTION.C.NS	33
3.2 TRAVELTIMEFUNCTION.C.SN	41
3.3 TRAVELTIMEFUNCTION.C.SN_EW	49
3.4 TRAVELTIMEFUNCTION.C.SN_WE	56

Chapter Five—AS-3

NOTE: Long code lines that do not fit completely on one line of this document are shown in italics and continued on to the next line.

1. CONFIGURATION FILES

1.1 allstr_actgen_river_cross.cfg

```

TRANSIMS_ROOT           /home/projects/transims/config/integration/scenarios/allstr
ACTIVITY_FILE           $TRANSIMS_ROOT/activity/activities

NET_DIRECTORY           $TRANSIMS_ROOT/network
NET_NODE_TABLE          Node.tbl
NET_LINK_TABLE           Link.tbl
NET_POCKET_LANE_TABLE   Pocket_Lane.tbl
NET_PARKING_TABLE        Parking.tbl
NET_LANE_CONNECTIVITY_TABLE Lane_Connectivity.tbl
NET_UNSIGNALIZED_NODE_TABLE Unsignalized_Node.tbl
NET_SIGNALIZED_NODE_TABLE Signalized_Node.tbl
NET_PHASEMING_PLAN_TABLE Phasing_Plan.tbl
NET_TIMING_PLAN_TABLE   Timing_Plan.tbl
NET_SPEED_TABLE          Speed.tbl
NET_LANE_USE_TABLE       Lane_Use.tbl
NET_TRANSIT_STOP_TABLE   Transit_Stop.tbl
NET_SIGNAL_COORDINATOR_TABLE Signal_Coordinator.tbl
NET_DETECTOR_TABLE       Detector.tbl
NET_TURN_PROHIBITION_TABLE Turn_Prohibition.tbl
NET_BARRIER_TABLE         Barrier.tbl
NET_ACTIVITY_LOCATION_TABLE Activity_Location.tbl
NET_PROCESS_LINK_TABLE   Process_Link.tbl

OUT_DIRECTORY            /home/Gershwinoutput1/allstr

ACT_HOME_HEADER          Home
ACT_WORK_HEADER           Work
ACT_ACCESS_HEADER          Tran_Dist
ACT_BLOCKGROUP_HEADER      BLOCKGROUP
ACT_TRACT_HEADER           TRACT
ACT_FEEDBACK_FILE          $TRANSIMS_ROOT/activity/activity.feedback
#ACT_MULTI_FAMILY_HEADER    MFR

```

ACT_PARTIAL_OUTPUT	\$TRANSIMS_ROOT/activity/activities.partial
ACT_POPULATION_FILE	\$TRANSIMS_ROOT/population/pop_converted
ACT_PROBLEM_FILE	\$TRANSIMS_ROOT/activity/act.problems
ACT_RANDOM_SEED	985456379
ACT_SHARED_RIDE_TIME_RANGE_MIN	1
ACT_SHARED_RIDE_TIME_RANGE_MEDIUM	1
ACT_SHARED_RIDE_TIME_RANGE_MAX	1
ACT_SHARED_RIDE_DISTANCE_RANGE	10
ACT_TRAVEL_TIME_INTERVALS_FILE	\$TRANSIMS_ROOT/data/time_intervals
ACT_TRAVEL_TIMES_FILE	\$TRANSIMS_ROOT/activity/travel_times
ACT_TRAVEL_TIMES_FILE	/home/Gershwinoutput1/dickb/traveltimes/traveltime_040601
ACT_TRAVEL_TIME_FUNCTION_MODES	1;2;3;7
ACT_MINIMUM_ADULT_AGE	16
ACT_DECISION_TREE_FILE	\$TRANSIMS_ROOT/data/decision_tree
ACT_MODE_WEIGHT_FILE	\$TRANSIMS_ROOT/data mode_weight
ACT_ZONE_INFO_FILE	\$TRANSIMS_ROOT/data/zonedata
ACT_SURVEY_ACTIVITY_FILE	\$TRANSIMS_ROOT/data/survey_activities
ACT_SURVEY_HOUSEHOLD_FILE	\$TRANSIMS_ROOT/data/survey_households
ACT_SURVEY_WEIGHTS_FILE	\$TRANSIMS_ROOT/data/survey_weights_4
ACT_INITIAL_HOME_TIME_RANGE	0.75
ACT_OUT_OF_HOME_TIME_RANGE	0.5
ACT_HOME_DURING_DAY_TIME_RANGE	0.75
ACT_WORK_TIME_RANGE	0.25
ACT_END_OF_DAY_TIME_RANGE	0.75
ACT_MAX_END_TIME	36.0
ACT_DEFAULT_CAR_SPEED	15.0
ACT_DEFAULT_TRANSIT_SPEED	10.0
ACT_BICYCLE_MODE	7
ACT_WALKING_MODE	1
ACT_MAGIC_MOVE_MODE	8
ACT_DEFAULT_TRANSIT_MODE	3
ACT_LOCATION_CHOICE_EXPONENT	0.5
ACT_DEFAULT_INTRAZONE_DISTANCE	1500
ACT_AUTOMOBILE_MODE_1	2
ACT_AUTOMOBILE_MODE_2	5
ACT_AUTOMOBILE_MODE_3	6
ACT_ADJUST_TIMES	0
ACT_PERSON_DEMOG_RELATION_HEADER	RELATE
ACT_PERSON_DEMOG_WORKER_HEADER	WORK
ACT_PERSON_DEMOG_GENDER_HEADER	GENDER
ACT_PERSON_DEMOG_AGE_HEADER	AGE
ACT_REQUIRED_HH_DEMOG_1	HHSIZE
ACT_REQUIRED_HH_DEMOG_2	INCOME
ACT_REQUIRED_HH_DEMOG_3	ALT5
ACT_REQUIRED_HH_DEMOG_4	A5TO15
ACT_REQUIRED_HH_DEMOG_5	A26TO45
ACT_REQUIRED_HH_DEMOG_6	HHAGE

```

ACT_REQUIRED_HH_DEMOG_7          WORKERS
ACT_REQUIRED_HH_DEMOG_8          HDENSTY
ACT_HHDENSITY_HEADER            HH_Acre

ACT_HOME_ACTIVITY_TYPE           0
ACT_WORK_ACTIVITY_TYPE           1
ACT_SCHOOL_ACTIVITY_TYPE         7

# School specifications: age ranges, zone and location attractor values.

# Elementary and mid school
ACT SCHOOL LOWER_BOUND_1          5
ACT SCHOOL UPPER_BOUND_1          15

# High school
ACT SCHOOL LOWER_BOUND_2          16
ACT SCHOOL UPPER_BOUND_2          18

# Elementary and high school
ACT SCHOOL LOWER_BOUND_3          5
ACT SCHOOL UPPER_BOUND_3          18

# Elementary and mid school
ACT SCHOOL_ZONE_ATTRACTOR_VALUE_1 1

# High school
ACT SCHOOL_ZONE_ATTRACTOR_VALUE_2 2

# Both elementary/mid and high school
ACT SCHOOL_ZONE_ATTRACTOR_VALUE_3 3

# Elementary and mid school
ACT SCHOOL_LOCATION_ATTRACTOR_VALUE_1 1

# High school
ACT SCHOOL_LOCATION_ATTRACTOR_VALUE_2 2

# Both elementary/mid and high school
ACT SCHOOL_LOCATION_ATTRACTOR_VALUE_3 3

# Anchor Activity types

# Home
ACT ANCHOR_ACTIVITY_TYPE_1        0

# Work
ACT ANCHOR_ACTIVITY_TYPE_2        1

# School
ACT ANCHOR_ACTIVITY_TYPE_3        7

```

# College	
ACT_ANCHOR_ACTIVITY_TYPE_4	8
ACT_ACTIVITY_TYPE_1	0
ACT_ZONE_HEADER_1	Home
ACT_LOCATION_HEADER_1	Home
ACT_PRIORITY_1	2
ACT_TIME_PRIORITY_1	3
ACT_ACTIVITY_TYPE_2	1
ACT_ZONE_HEADER_2	Work
ACT_LOCATION_HEADER_2	Work
ACT_PRIORITY_2	2
ACT_TIME_PRIORITY_2	3
ACT_ACTIVITY_TYPE_3	2
ACT_ZONE_HEADER_3	Shop
ACT_LOCATION_HEADER_3	Shop
ACT_PRIORITY_3	7
ACT_TIME_PRIORITY_3	0
ACT_ACTIVITY_TYPE_4	3
ACT_ZONE_HEADER_4	Visit
ACT_LOCATION_HEADER_4	Visit
ACT_PRIORITY_4	4
ACT_TIME_PRIORITY_4	3
ACT_ACTIVITY_TYPE_5	4
ACT_ZONE_HEADER_5	Social
ACT_LOCATION_HEADER_5	Social
ACT_PRIORITY_5	4
ACT_TIME_PRIORITY_5	5
ACT_ACTIVITY_TYPE_6	5
ACT_ZONE_HEADER_6	Other
ACT_LOCATION_HEADER_6	Other
ACT_PRIORITY_6	8
ACT_TIME_PRIORITY_6	0
ACT_ACTIVITY_TYPE_7	6
ACT_ZONE_HEADER_7	Serve
ACT_LOCATION_HEADER_7	Serve
ACT_PRIORITY_7	4
ACT_TIME_PRIORITY_7	1
ACT_ACTIVITY_TYPE_8	7
ACT_ZONE_HEADER_8	School
ACT_LOCATION_HEADER_8	School
ACT_PRIORITY_8	3
ACT_TIME_PRIORITY_8	3

ACT_ACTIVITY_TYPE_9	8
ACT_ZONE_HEADER_9	College
ACT_LOCATION_HEADER_9	College
ACT_PRIORITY_9	3
ACT_TIME_PRIORITY_9	3
ACT_TAZ_HEADER	TAZ
# Selector Keys	
SEL_MESSAGE_LEVEL	0
SEL_NO_ITDB_INDEX	1
SEL_USE_END_ACT_LOCATION	1
SEL_USE_END_ACT_TYPE	1
SEL_USE_END_MODE_PREF	1
SEL_USE_END_OTHER_PARTICIPANTS	1
#SEL_USE_START_ACT_USER_DATA	River_Zone
SEL_USE_END_ACT_USER_DATA	River_Zone
SEL_USE_DRIVES_PASSENGER	1

2. SCRIPTS

2.1 SplitActByZone.pl

```

#!/usr/local/bin/perl -w

# split activities according to river zone household is located in
# $ARGV[0] is activity location table
# $ARGV[1] is activity file
# $ARGV[2] is activities for households with home in north
# $ARGV[3] is activities for households with home in west
# $ARGV[4] is activities for households with home in east

use English;

# read activity location file and store river zone for each location
open IN, "<$ARGV[0]" or die "Couldn't open $ARGV[0] for input ($OS_ERROR)\n";
$line = <IN>;
while (<IN>)
{
    chomp;
    @line = split;
    $id = $line[0];
    $rzones{$id} = $line[24];
}
close IN;
$nrz = keys %rzones;
print "size of activity location array is $nrz\n";

$hhid = -1;
$first = 1;
open IN, "<$ARGV[1]" or die "Couldn't open $ARGV[1] for input ($OS_ERROR)\n";
open N, ">$ARGV[2]" or die "Couldn't open $ARGV[2] for output ($OS_ERROR)\n";
open W, ">$ARGV[3]" or die "Couldn't open $ARGV[4] for output ($OS_ERROR)\n";
open E, ">$ARGV[4]" or die "Couldn't open $ARGV[6] for output ($OS_ERROR)\n";
open JUNK, ">junksplit" or die "Couldn't open junksplit for output ($OS_ERROR)\n";
while (<IN>)
{
    chomp;
    @line = split;
    If ($line[0] == $hhid)      # same household
    {
        push @hh, [ @line ];
        next;
    }
    if (!$first) # analyze complete household

```

```

{
    AnalyzeHousehold();
}

# begin new household
$hhid = $line[0];
@hh = ();
push @hh, [ @line ];
$first = 0;
}
if (!$first) { AnalyzeHousehold(); }

sub AnalyzeHousehold
{
    $hreg = $rzones{$hh[0][20]};
    if ($hreg == 1) { $FH = *N; }
    elsif ($hreg == 2) { $FH = *W; }
    elsif ($hreg == 3) { $FH = *E; }
    else
    {
        print "household $hhid in region $hreg\n";
#        return;
        $FH = *JUNK;
    }
    for $i (0 .. $#hh)
    {
        $aref = $hh[$i];
        for $j (0 .. $#{$aref})
        {
            print $FH "$aref->[$j]\t";
        }
        print $FH "\n" or die "Can't write $hhid to $FH ($OS_ERROR)\n";
    }
}
}

```

2.2 FindCrossings.pl

```

#!/usr/local/bin/perl -w

# find river crossings in itdb file
# ARGV[3] is the tour type: NSW, NSS, NSO, SNW, SNS, SNO, EW, WE, ALL
# ARGV[4] indicates whether to save tours in feedback file

use English;

open IN, "<$ARGV[0]" or die "Couldn't open $ARGV[0] for input ($OS_ERROR)\n";
open OUT, ">$ARGV[1]" or die "Couldn't open $ARGV[1] for output ($OS_ERROR)\n";
open LOG, ">>$ARGV[2]" or die "Couldn't open $ARGV[2] for output ($OS_ERROR)\n";
print LOG "tour type is $ARGV[3]\n";

%counts = ();
$counts{ "ALL" } = 0;
$counts{ "NSW" } = 0;
$counts{ "NSS" } = 0;
$counts{ "NSO" } = 0;
$counts{ "SNW" } = 0;
$counts{ "SNS" } = 0;
$counts{ "SNO" } = 0;
$counts{ "EWW" } = 0;
$counts{ "EWS" } = 0;
$counts{ "EWO" } = 0;
$counts{ "WEW" } = 0;
$counts{ "WES" } = 0;
$counts{ "WEO" } = 0;
$counts{ "NSWP" } = 0;
$counts{ "NSSP" } = 0;
$counts{ "NSOP" } = 0;
$counts{ "SNWP" } = 0;
$counts{ "SNSP" } = 0;
$counts{ "SNOP" } = 0;
$counts{ "EWWP" } = 0;
$counts{ "EWSP" } = 0;
$counts{ "EWOP" } = 0;
$counts{ "WEWP" } = 0;
$counts{ "WESP" } = 0;
$counts{ "WEOP" } = 0;
$total = 0;
$tourid = -1;
$first = 1;
# skip header lines
$line = <IN>;
$line = <IN>;
while (<IN>)

```

```

{
    chomp;
#   0      1      2      3      4      5      6      7      8      9      10     11      12
# ($hh, $trav, $tour, $subtour, $strip, $sid, $eid, $etype, $emode, $eloc, $eother, $drivespass, $ereg)
@line = split //,/;
if ($line[2] == $tourid) # same tour
{
    push @tour, [ @line ];
    next;
}

if (!$first) # analyze complete tour
{
    AnalyzeTour();
    $first = 1;
}

# begin new tour
if ($line[2] == 0)      # home activity
{
    $home = $line[9];
    $hreg = $line[12];
    $tourid = -1;
}
else                      # next tour
{
    $tourid = $line[2];
    @tour = ();
    push @tour, [ @line ];
    $first = 0;
}
}
if (!$first) { AnalyzeTour(); }

if ($ARGV[3] eq 'ALL' || $ARGV[4])
{
    print LOG qq!Totals: N-S work tours = $counts{"NSW"}\tN-S shop tours = $counts{"NSS"}\tN-S other tours = $counts{"NSO"}\n!;
    print LOG qq!Shared: N-S work tours = $counts{"NSWP"}\tN-S shop tours = $counts{"NSSP"}\tN-S other tours = $counts{"NSOP"}\n!;
    print LOG qq!Totals: S-N work tours = $counts{"SNW"}\tS-N shop tours = $counts{"SNS"}\tS-N other tours = $counts{"SNO"}\n!;
    print LOG qq!Shared: S-N work tours = $counts{"SNWP"}\tS-N shop tours = $counts{"SNSP"}\tS-N other tours = $counts{"SNOP"}\n!;
    print LOG qq!Totals: W-E work tours = $counts{"WEW"}\tW-E shop tours = $counts{"WES"}\tW-E other tours = $counts{"WEO"}\n!;
    print LOG qq!Shared: W-E work tours = $counts{"WEWP"}\tW-E shop tours = $counts{"WESP"}\tW-E other tours = $counts{"WEOP"}\n!;
    print LOG qq!Totals: E-W work tours = $counts{"EWW"}\tE-W shop tours = $counts{"EWS"}\tE-W other tours = $counts{"EWO"}\n!;
    print LOG qq!Shared: E-W work tours = $counts{"EWWP"}\tE-W shop tours = $counts{"EWSP"}\tE-W other tours = $counts{"EWOP"}\n!;
    print LOG "Total tours = $total\n";
}
elsif ($ARGV[3] eq 'NSW') { print LOG qq!N-S work tours = $counts{"NSW"}\tN-S work tours shared = $counts{"NSWP"}\n!; }
elsif ($ARGV[3] eq 'NSS') { print LOG qq!N-S shop tours = $counts{"NSS"}\tN-S shop tours shared = $counts{"NSSP"}\n!; }
elsif ($ARGV[3] eq 'NSO') { print LOG qq!N-S other tours = $counts{"NSO"}\tN-S other tours shared = $counts{"NSOP"}\n!; }
elsif ($ARGV[3] eq 'SNW') { print LOG qq!S-N work tours = $counts{"SNW"}\tS-N work tours shared = $counts{"SNWP"}\n!; }
elsif ($ARGV[3] eq 'SNS') { print LOG qq!S-N shop tours = $counts{"SNS"}\tS-N shop tours shared = $counts{"SNSP"}\n!; }

```

```

elsif ($ARGV[3] eq 'SNO') { print LOG qq!S-N other tours = $counts{"SNO"}\tS-N other tours shared = $counts{"SNOP"}\n!; }
elsif ($ARGV[3] eq 'WEW') { print LOG qq!W-E work tours = $counts{"NEW"}\tW-E work tours shared = $counts{"WEWP"}\n!; }
elsif ($ARGV[3] eq 'WES') { print LOG qq!W-E shop tours = $counts{"WES"}\tW-E shop tours shared = $counts{"WESP"}\n!; }
elsif ($ARGV[3] eq 'WEO') { print LOG qq!W-E other tours = $counts{"WEO"}\tW-E other tours shared = $counts{"WEOP"}\n!; }
elsif ($ARGV[3] eq 'EWW') { print LOG qq!E-W work tours = $counts{"EWW"}\tE-W work tours shared = $counts{"EWWP"}\n!; }
elsif ($ARGV[3] eq 'EWS') { print LOG qq!E-W shop tours = $counts{"EWS"}\tE-W shop tours shared = $counts{"EWSP"}\n!; }
elsif ($ARGV[3] eq 'EWO') { print LOG qq!E-W other tours = $counts{"EWO"}\tE-W other tours shared = $counts{"EWOP"}\n!; }
for $key (keys %counts)
{
    if ($key eq $ARGV[3]) { $tmp = $counts{$key}; }
}
if ($ARGV[3] ne 'ALL') { print LOG "$tmp\n"; }

sub AnalyzeTour
{
    $total++;
    $hhid = $tour[0][0];
    $sactid = $tour[0][5];
    $work = $shop = $shared = $car = $worktrip = $drivespass = 0;
    $crossns = $crosssn = $crossew = $crosswe = 0;
    $crossnsw = $crossnw = $crossww = $crosswew = 0;
    for ($i=0; $i < $#tour; $i++)
    {
        if ($tour[$i][9] eq $home) { next; }
        if ($tour[$i][10] > 0) { $shared = 1; }
        if ($tour[$i][11] eq "true") { $drivespass = 1; }
        if ($tour[$i][7] eq 1) { $work = 1; $worktrip = $i; }
        if ($tour[$i][7] eq 8) { $work = 1; $worktrip = $i; }
        if ($tour[$i][7] eq 2) { $shop = 1; }
        if ($tour[$i][8] eq 2) { $car = 1; }
        if ($shreg eq 1 && $tour[$i][12] ne 1) { $crossns = 1; }
        if ($shreg ne 1 && $tour[$i][12] eq 1) { $crosssn = 1; }
        if ($shreg eq 2 && $tour[$i][12] eq 3) { $crosswe = 1; }
        if ($shreg eq 3 && $tour[$i][12] eq 2) { $crossew = 1; }
        if ($shreg eq 1 && $tour[$i][12] ne 1 && $worktrip eq $i) { $crossnsw = 1; }
        if ($shreg ne 1 && $tour[$i][12] eq 1 && $worktrip eq $i) { $crossnw = 1; }
        if ($shreg eq 2 && $tour[$i][12] eq 3 && $worktrip eq $i) { $crosswew = 1; }
        if ($shreg eq 3 && $tour[$i][12] eq 2 && $worktrip eq $i) { $crossww = 1; }
    }
    if ($work && $shop) { $shop = 0; }
    if ($crossns && $car)
    {
        if ($work && $crossnsw) { $counts{"NSW"}++; }
        elsif ($shop) { $counts{"NSS"}++; }
        elsif (!$work && !$shop) { $counts{"NSO"}++; }
        if ($shared && $work && $crossnsw) { $counts{"NSWP"}++; }
        elsif ($shared && $shop) { $counts{"NSSP"}++; }
        elsif ($shared && !$work && !$shop) { $counts{"NSOP"}++; }
    }
    elsif ($crosssn && $car)

```

```

{
    if ($work && $crosssnw) { $counts{"SNW"}++; }
    elsif ($shop) { $counts{"SNS"}++; }
    elsif (!$work && !$shop) { $counts{"SNO"}++; }
    if ($shared && $work && $crosssnw) { $counts{"SNWP"}++; }
    elsif ($shared && $shop) { $counts{"SNSP"}++; }
    elsif ($shared && !$work && !$shop) { $counts{"SNOP"}++; }
}
elsif ($crosswe && $car)
{
    if ($work && $crosswew) { $counts{"WEW"}++; }
    elsif ($shop) { $counts{"WES"}++; }
    elsif (!$work && !$shop) { $counts{"WEO"}++; }
    if ($shared && $work && $crosswew) { $counts{"WEWP"}++; }
    elsif ($shared && $shop) { $counts{"WESP"}++; }
    elsif ($shared && !$work && !$shop) { $counts{"WEOP"}++; }
}
elsif ($crossew && $car)
{
    if ($work && $crosseww) { $counts{"EWW"}++; }
    elsif ($shop) { $counts{"EWS"}++; }
    elsif (!$work && !$shop) { $counts{"EWO"}++; }
    if ($shared && $work && $crosseww) { $counts{"EWWP"}++; }
    elsif ($shared && $shop) { $counts{"EWP"}++; }
    elsif ($shared && !$work && !$shop) { $counts{"EWOP"}++; }
}
if ($ARGV[4])
{
if ($ARGV[3] eq "NSW" && $hreg eq 1 && $work && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "NSS" && $hreg eq 1 && $shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "NSO" && $hreg eq 1 && !$work && !$shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "SNW" && $hreg ne 1 && $work && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "SNS" && $hreg ne 1 && $shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "SNO" && $hreg ne 1 && !$work && !$shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "WEW" && $hreg eq 2 && $work && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "WES" && $hreg eq 2 && $shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "WEO" && $hreg eq 2 && !$work && !$shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "EWW" && $hreg eq 3 && $work && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "EWS" && $hreg eq 3 && $shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
elsif ($ARGV[3] eq "EWO" && $hreg eq 3 && !$work && !$shop && (!$shared || $drivespass))
    { print OUT "$hhid $sactid LTR $sactid\n"; }
}

```

```
        { print OUT "$hhid $sactid LTR $sactid\n"; }
}
```

2.3 FixBridgeCross.NS

```

#! /bin/csh

# The following 5 parameters have no default value
# $1 is tour type, e.g. NSW, NSS, NSO, SNW, SNS, SNO, EW, WE
# $2 is target number of crossings
# $3 is initial value of c
# $4 is activity file
# $5 is number of processors to use

# The following 4 parameters are for continuation of a previous non-converged run
# $6 is previous value of c for which result was above target, or "none" if none known
# $7 is previous value of c for which result was below target, or "none" if none known
# $8 is iteration number to begin with
# $9 is iteration number to end at

setenv SCENARIO $TRANSIMS_HOME/scenarios/allstr
setenv CONFIG_FILE $SCENARIO/config_files/allstr_actgen_river_cross.cfg
setenv LOG_FILE $SCENARIO/log/ActRegenCross$1.log
setenv ACT_LOG_FILE $SCENARIO/log/ActRegen$1.log
setenv ACT_DIR $SCENARIO/activity
setenv SCRIPT_DIR $ACT_DIR
setenv ACTIVITY_FILE $ACT_DIR/$4
setenv PARTIAL_FILE $ACT_DIR/cpar$1
setenv PROBLEM_FILE $ACT_DIR/cprob$1
setenv ITDB_FILE $ACT_DIR/itcross$1
setenv FEEDBACK_COMMANDS $ACT_DIR/fbcross$1
setenv HOUSEHOLD_FILE $ACT_DIR/hhcross$1
setenv TMP_CONFIG $ACT_DIR/cross$1.cfg

setenv MERGE_INDEX $TRANSIMS_HOME/bin/MergeIndices
setenv INDEXDEFRAG $TRANSIMS_HOME/bin/IndexDefrag
setenv COLLATOR $TRANSIMS_HOME/bin/Collator
setenv ACTREGEN $TRANSIMS_HOME/bin/ActivityRegenerator
setenv HOUSEHOLDS $TRANSIMS_HOME/bin/MakeHouseholdFile
setenv DTOA $TRANSIMS_HOME/bin/10to26
setenv INDEX_ACT $TRANSIMS_HOME/bin/IndexActivityFile
setenv PERL /usr/bin/perl
setenv GAWK /usr/bin/gawk

@ targetlb = ($2 * 95) / 100
@ targetub = ($2 * 105) / 100
@ c = 0
set above = 0
set below = 0
set istart = 1
set iend = 1

```

```

set mxavail = `tail -100 $ACT_DIR/machines.$1`
@ mstart = 0
set length = `wc -l $ACT_DIR/machines.$1`
@ mend = $length[1]
@ muse = $5

# Check for continuation run
if ($#argv > 5) then
    echo "Continuation of previous iteration" >>&! $LOG_FILE
    @ c = $3
    if ($6 != "none") then
        set above = 1
        @ cmin = $6
    endif
    if ($7 != "none") then
        set below = 1
        @ cmax = $7
    endif
    set istart = $8
    set iend = $9
    goto loop
endif

# Regenerate the activities that crossed the river
echo "Regenerating activities to remove river crossings..." >>&! $LOG_FILE
echo "Target range for crossings is" $targetlb "to" $targetub >>&! $LOG_FILE

/bin/cp $CONFIG_FILE $TMP_CONFIG
/bin/chmod u+w $TMP_CONFIG

echo "ROUTER_HOUSEHOLD_FILE " $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "SEL_HOUSEHOLD_FILE " $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "ACTIVITY_FILE " $ACTIVITY_FILE >> $TMP_CONFIG
echo "SEL_ITDB_FILE " $ITDB_FILE >> $TMP_CONFIG
echo "ACT_FEEDBACK_FILE " $FEEDBACK_COMMANDS >> $TMP_CONFIG
echo "ACT_LOG_FILE " $ACT_LOG_FILE >> $TMP_CONFIG

echo "Dividing households among $muse processors" >>&! $LOG_FILE
$HOUSEHOLDS $TMP_CONFIG $muse >>&! $LOG_FILE

echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $ACTIVITY_FILE >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then

```

```

if ($m >= $mstart && $m <= $mend && $mused < $muse) then
    set mxused = ($mxused $mx)
    @ mused++
endif
@ m++
else
    echo "No response from $mx"
endif
end
# Spawn processes
@ m = 0
echo "Running Collator" >>&! $LOG_FILE
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
    @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.000.it >! $ITDB_FILE.000.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.000.it >> $ITDB_FILE.000.it
    @ m++
end
echo "Finding river crossings in iteration database 0 " >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.000.it $FEEDBACK_COMMANDS $LOG_FILE $1 1
@ cross = `tail -1 $LOG_FILE`
if ($cross > $targetub) then
    echo $cross "is too many crossings" >>&! $LOG_FILE
    set above = 1
    @ cmin = 0
    @ c = $3
else if ($cross < $targetlb) then
    echo $cross "is too few crossings" >>&! $LOG_FILE
    set below = 1

```

```

@ cmax = 0
@ c = $3
else
    echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
    echo "Finished." >>& $LOG_FILE
    exit 0
endif

loop:
set done = 0
# Do iterations
set i = $istart
while ($i <= $iend)
    echo "Running Activity Regenerator for iteration" $i "with c =" $c >>&! $LOG_FILE
    echo "ACT_PARTIAL_OUTPUT " $PARTIAL_FILE.$i >> $TMP_CONFIG
    echo "ACT_PROBLEM_FILE " $PROBLEM_FILE.$i >> $TMP_CONFIG
    echo "ACT_TRAVEL_TIME_FUNCTION_PARAMETERS " $c >> $TMP_CONFIG
    $ACTREGEN $TMP_CONFIG >>&! $LOG_FILE

    # remove indices created by ActivityRegenerator because sometimes they are corrupted
    /bin/rm -f $PARTIAL_FILE.$i.*.idx
    echo "Indexing activity file" >>&! $LOG_FILE
    $INDEX_ACT $PARTIAL_FILE.$i >>&! $LOG_FILE
    sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then
        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
            set mxused = ($mxused $mx)
            @ mused++
        endif
        @ m++
    else
        echo "No response from $mx"
    endif
end
echo "ACTIVITY_FILE " $PARTIAL_FILE.$i >> $TMP_CONFIG
sleep 10
echo "Running Collator" >>&! $LOG_FILE
# Spawn processes
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix

```

```

        @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.00$i.it >! $ITDB_FILE.00$i.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.00$i.it >> $ITDB_FILE.00$i.it
    @ m++
end

echo "Finding river crossings in iteration database" $i >&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.00$i.it crossjunk$1 $LOG_FILE $1 0
@ cross = `tail -1 $LOG_FILE`
if ($cross > $targetub) then
    echo $cross "is too many crossings" >&! $LOG_FILE
    set above = 1
    @ cmin = $c
    if ($below == 1) then
        @ c = ($cmin + $cmax) / 2
    else
        @ c = 2 * $c
    endif
else if ($cross < $targetlb) then
    echo $cross "is too few crossings" >&! $LOG_FILE
    set below = 1
    @ cmax = $c
    if ($above == 1) then
        @ c = ($cmin + $cmax) / 2
    else
        @ c = 2 * $c
    endif
else
    echo $cross "is an acceptable number of crossings with c =" $c >&! $LOG_FILE
    set done = $i
    break
endif
set i = `expr $i + 1`
end

```

```
if ($done == 0) then
    echo "Iterations completed without converging to target" >>&! $LOG_FILE
    echo "New activity file not created" >>&! $LOG_FILE
    echo " " >>&! $LOG_FILE
    exit 1
endif

# Merge the partial activity set indexes with the original activity set index.
echo "Merging partial activity indexes.. " >>& $LOG_FILE
$MERGE_INDEX $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.hh.idx $PARTIAL_FILE.$done.hh.idx >>&! $LOG_FILE

# Create a merged activity file from the merged indexes
echo "Creating merged activity file.." >>& $LOG_FILE
$INDEXDEFRAG $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.$1 >>&! $LOG_FILE

echo "Finished." >>& $LOG_FILE
```

2.4 FixBridgeCross.SN

```

#! /bin/csh

# The following 5 parameters have no default value
# $1 is tour type, e.g. NSW, NSS, NSO, SNW, SNS, SNO, EW, WE
# $2 is target number of crossings
# $3 is initial value of c
# $4 is activity file
# $5 is number of processors to use

# The following 4 parameters are for continuation of a previous non-converged run
# $6 is previous value of c for which result was above target, or "none" if none known
# $7 is previous value of c for which result was below target, or "none" if none known
# $8 is iteration number to begin with
# $9 is iteration number to end at

setenv SCENARIO $TRANSIMS_HOME/scenarios/allstr
setenv CONFIG_FILE $SCENARIO/config_files/allstr_actgen_river_cross.cfg
setenv LOG_FILE $SCENARIO/log/ActRegenCross$1.log
setenv ACT_LOG_FILE $SCENARIO/log/ActRegen$1.log
setenv ACT_DIR $SCENARIO/activity
setenv SCRIPT_DIR $ACT_DIR
setenv ACTIVITY_FILE $ACT_DIR/$4
setenv PARTIAL_FILE $ACT_DIR/cpar$1
setenv PROBLEM_FILE $ACT_DIR/cprob$1
setenv ITDB_FILE $ACT_DIR/itcross$1
setenv FEEDBACK_COMMANDS $ACT_DIR/fbcross$1
setenv HOUSEHOLD_FILE $ACT_DIR/hhcross$1
setenv TMP_CONFIG $ACT_DIR/cross$1.cfg

setenv MERGE_INDEX $TRANSIMS_HOME/bin/MergeIndices
setenv INDEXDEFRAG $TRANSIMS_HOME/bin/IndexDefrag
setenv COLLATOR $TRANSIMS_HOME/bin/Collator
setenv ACTREGEN $TRANSIMS_HOME/bin/ActivityRegenerator
setenv HOUSEHOLDS $TRANSIMS_HOME/bin/MakeHouseholdFile
setenv DTOA $TRANSIMS_HOME/bin/10to26
setenv INDEX_ACT $TRANSIMS_HOME/bin/IndexActivityFile
setenv PERL /usr/bin/perl
setenv GAWK /usr/bin/gawk

@ targetlb = ($2 * 95) / 100
@ targetub = ($2 * 105) / 100
@ c = 0
set above = 0
set below = 0
set istart = 1

```

```

set iend = 1
set mxavail = `tail -100 $ACT_DIR/machines.$1`
@ mstart = 0
set length = `wc -l $ACT_DIR/machines.$1`
@ mend = $length[1]
@ muse = $5

# Check for continuation run
if ($#argv > 5) then
    echo "Continuation of previous iteration" >>&! $LOG_FILE
    @ c = $3
    if ($6 != "none") then
        set above = 1
        @ cmin = $6
    endif
    if ($7 != "none") then
        set below = 1
        @ cmax = $7
    endif
    set istart = $8
    set iend = $9
    goto loop
endif

# Regenerate the activities that crossed the river
echo "Regenerating activities to remove river crossings..." >>&! $LOG_FILE
echo "Target range for crossings is" $targetlb "to" $targetub >>&! $LOG_FILE

/bin/cp $CONFIG_FILE $TMP_CONFIG
/bin/chmod u+w $TMP_CONFIG

echo "ROUTER_HOUSEHOLD_FILE " $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "SEL_HOUSEHOLD_FILE " $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "ACTIVITY_FILE " $ACTIVITY_FILE >> $TMP_CONFIG
echo "SEL_ITDB_FILE " $ITDB_FILE >> $TMP_CONFIG
echo "ACT_FEEDBACK_FILE " $FEEDBACK_COMMANDS >> $TMP_CONFIG
echo "ACT_LOG_FILE " $ACT_LOG_FILE >> $TMP_CONFIG

echo "Dividing households among $muse processors" >>&! $LOG_FILE
$HOUSEHOLDS $TMP_CONFIG $muse >>&! $LOG_FILE

echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $ACTIVITY_FILE >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`


```

```

if ("X$result" == "X") then
    if ($m >= $mstart && $m <= $mend && $mused < $muse) then
        set mxused = ($mxused $mx)
        @ mused++
    endif
    @ m++
else
    echo "No response from $mx"
endif
end
# Spawn processes
@ m = 0
echo "Running Collator" >>&! $LOG_FILE
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
    @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.000.it >! $ITDB_FILE.000.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.000.it >> $ITDB_FILE.000.it
    @ m++
end

echo "Finding river crossings in iteration database 0 " >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.000.it $FEEDBACK_COMMANDS $LOG_FILE $1 1
@ cross = `tail -1 $LOG_FILE`
if ($cross > $targetub) then
    echo $cross "is too many crossings" >>&! $LOG_FILE
    set above = 1
    @ cmin = 0
    @ c = $3
else if ($cross < $targetlb) then
    echo $cross "is too few crossings" >>&! $LOG_FILE

```

```

set below = 1
@ cmax = 0
@ c = $3
else
echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
echo "Finished." >>& $LOG_FILE
exit 0
endif

loop:
set done = 0
# Do iterations
set i = $istart
while ($i <= $iend)
echo "Running Activity Regenerator for iteration" $i "with c =" $c >>&! $LOG_FILE
echo "ACT_PARTIAL_OUTPUT" $PARTIAL_FILE.$i >> $TMP_CONFIG
echo "ACT_PROBLEM_FILE" $PROBLEM_FILE.$i >> $TMP_CONFIG
echo "ACT_TRAVEL_TIME_FUNCTION_PARAMETERS" $c >> $TMP_CONFIG
$ACTREGEN $TMP_CONFIG >>&! $LOG_FILE

# remove indices created by ActivityRegenerator because sometimes they are corrupted
/bin/rm -f $PARTIAL_FILE.$i.*.idx
echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $PARTIAL_FILE.$i >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then
        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
            set mxused = ($mxused $mx)
            @ mused++
        endif
        @ m++
    else
        echo "No response from $mx"
    endif
end
echo "ACTIVITY_FILE" $PARTIAL_FILE.$i >> $TMP_CONFIG
sleep 10
echo "Running Collator" >>&! $LOG_FILE
# Spawn processes
@ m = 0
foreach mx ($mxused)
    set suffix = `DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE

```

```

ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
@ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.00$i.it >! $ITDB_FILE.00$i.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.00$i.it >> $ITDB_FILE.00$i.it
    @ m++
end

echo "Finding river crossings in iteration database" $i >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.00$i.it crossjunk$1 $LOG_FILE $1 0
@ cross = `tail -1 $LOG_FILE`
if ($cross > $targetub) then
    echo $cross "is too many crossings" >>&! $LOG_FILE
    set above = 1
    @ cmin = $c
    if ($below == 1) then
        @ c = ($cmin + $cmax) / 2
    else
        @ c = 2 * $c
    endif
else if ($cross < $targetlb) then
    echo $cross "is too few crossings" >>&! $LOG_FILE
    set below = 1
    @ cmax = $c
    if ($above == 1) then
        @ c = ($cmin + $cmax) / 2
    else
        @ c = 2 * $c
    endif
else
    echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
    set done = $i
    break
endif
set i = `expr $i + 1`
```

```
end

if ($done == 0) then
    echo "Iterations completed without converging to target" >>&! $LOG_FILE
    echo "New activity file not created" >>&! $LOG_FILE
    echo " " >>&! $LOG_FILE
    exit 1
endif

echo "Finished." >>& $LOG_FILE
```

2.5 FixBridgeCross.WE

```

#!/bin/csh

# The following parameters have no default value
# $1 is tour type, e.g. WEW, WES, WEO
# $2 is correction for SN* crossings
# $3 is initial value of c
# $4 is activity file
# $5 is number of processors to use
# $6 is iteration number

setenv SCENARIO $TRANSIMS_HOME/scenarios/allstr
setenv CONFIG_FILE $SCENARIO/config_files/allstr_actgen_river_cross.cfg
setenv LOG_FILE $SCENARIO/log/ActRegenCross$1.log
setenv ACT_LOG_FILE $SCENARIO/log/ActRegen$1.log
setenv ACT_DIR $SCENARIO/activity
setenv SCRIPT_DIR $ACT_DIR
setenv ACTIVITY_FILE $ACT_DIR/$4
setenv PARTIAL_FILE $ACT_DIR/cpar$1
setenv PROBLEM_FILE $ACT_DIR/cprob$1
setenv ITDB_FILE $ACT_DIR/itcross$1
setenv FEEDBACK_COMMANDS $ACT_DIR/fbcross$1
setenv HOUSEHOLD_FILE $ACT_DIR/hhcross$1
setenv TMP_CONFIG $ACT_DIR/cross$1.cfg

setenv MERGE_INDEX $TRANSIMS_HOME/bin/MergeIndices
setenv INDEXDEFRAG $TRANSIMS_HOME/bin/IndexDefrag
setenv COLLATOR $TRANSIMS_HOME/bin/Collator
setenv ACTREGEN $TRANSIMS_HOME/bin/ActivityRegenerator.tt2
setenv HOUSEHOLDS $TRANSIMS_HOME/bin/MakeHouseholdFile
setenv DTOA $TRANSIMS_HOME/bin/10to26
setenv INDEX_ACT $TRANSIMS_HOME/bin/IndexActivityFile
setenv PERL /usr/bin/perl
setenv GAWK /usr/bin/gawk

set mxavail = `tail -100 $ACT_DIR/machines.$1`
@ mstart = 0
set length = `wc -l $ACT_DIR/machines.$1`
@ mend = $length[1]
@ muse = $5
@ c = $3
@ cc = $2

# check for continuation run
if ($6 > 1) then
    echo "Continuation of previous iteration" >>&! $LOG_FILE

```

```

        goto loop
endif

# Regenerate the activities that crossed the river
echo "Regenerating activities to remove river crossings..." >>&! $LOG_FILE

/bin/cp $CONFIG_FILE $TMP_CONFIG
/bin/chmod u+w $TMP_CONFIG

echo "ROUTER_HOUSEHOLD_FILE " $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "SEL_HOUSEHOLD_FILE " $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "ACTIVITY_FILE " $ACTIVITY_FILE >> $TMP_CONFIG
echo "SEL_ITDB_FILE " $ITDB_FILE >> $TMP_CONFIG
echo "ACT_FEEDBACK_FILE " $FEEDBACK_COMMANDS >> $TMP_CONFIG
echo "ACT_LOG_FILE " $ACT_LOG_FILE >> $TMP_CONFIG

echo "Dividing households among $muse processors" >>&! $LOG_FILE
$HOUSEHOLDS $TMP_CONFIG $muse >>&! $LOG_FILE

echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $ACTIVITY_FILE >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then
        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
            set mxused = ($mxused $mx)
            @ mused++
        endif
        @ m++
    else
        echo "No response from $mx"
    endif
end
# Spawn processes
@ m = 0
echo "Running Collator" >>&! $LOG_FILE
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
    @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator

```

```

sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.000.it >! $ITDB_FILE.000.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.000.it >> $ITDB_FILE.000.it
    @ m++
end

echo "Finding river crossings in iteration database 0 " >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.000.it $FEEDBACK_COMMANDS $LOG_FILE $1 1

loop:
set i = $6
echo "Running Activity Regenerator for iteration" $i "with c =" $c >>&! $LOG_FILE
echo "ACT_PARTIAL_OUTPUT" $PARTIAL_FILE.$i >> $TMP_CONFIG
echo "ACT_PROBLEM_FILE" $PROBLEM_FILE.$i >> $TMP_CONFIG
echo "ACT_TRAVEL_TIME_FUNCTION_PARAMETERS" $cc\$c >> $TMP_CONFIG
$ACTREGEN $TMP_CONFIG >>&! $LOG_FILE

# remove indices created by ActivityRegenerator because sometimes they are corrupted
/bin/rm -f $PARTIAL_FILE.$i.*.idx
echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $PARTIAL_FILE.$i >>&! $LOG_FILE
sleep 10
# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then
        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
            set mxused = ($mxused $mx)
            @ mused++
        endif
        @ m++
    else
        echo "No response from $mx"
    endif
end

```

```

echo "ACTIVITY_FILE" $PARTIAL_FILE.$i >> $TMP_CONFIG
sleep 10
echo "Running Collator" >>&! $LOG_FILE
# Spawn processes
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
    @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
    @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.00$i.it >! $ITDB_FILE.00$i.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.00$i.it >> $ITDB_FILE.00$i.it
    @ m++
end

echo "Finding river crossings in iteration database" $i >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.00$i.it crossjunk$1 $LOG_FILE $1 0

# Merge the partial activity set indexes with the original activity set index.
echo "Merging partial activity indexes.. " >>& $LOG_FILE
$MERGE_INDEX $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.hh.idx $PARTIAL_FILE.$i.hh.idx >>&! $LOG_FILE

# Create a merged activity file from the merged indexes
echo "Creating merged activity file.." >>& $LOG_FILE
$INDEXDEFrag $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.$1 >>&! $LOG_FILE

echo "Finished iteration $i." >>& $LOG_FILE

```

2.6 FixBridgeCross.EW

```
#!/bin/csh

# The following parameters have no default value
# $1 is tour type, e.g. EWW, EWS, EWO
# $2 is correction for SN* crossings
# $3 is initial value of c
# $4 is activity file
# $5 is number of processors to use
# $6 is iteration number

setenv SCENARIO $TRANSIMS_HOME/scenarios/allstr
setenv CONFIG_FILE $SCENARIO/config_files/allstr_actgen_river_cross.cfg
setenv LOG_FILE $SCENARIO/log/ActRegenCross$1.log
setenv ACT_LOG_FILE $SCENARIO/log/ActRegen$1.log
setenv ACT_DIR $SCENARIO/activity
setenv SCRIPT_DIR $ACT_DIR
setenv ACTIVITY_FILE $ACT_DIR/$4
setenv PARTIAL_FILE $ACT_DIR/cpar$1
setenv PROBLEM_FILE $ACT_DIR/cprob$1
setenv ITDB_FILE $ACT_DIR/itcross$1
setenv FEEDBACK_COMMANDS $ACT_DIR/fbcross$1
setenv HOUSEHOLD_FILE $ACT_DIR/hhcross$1
setenv TMP_CONFIG $ACT_DIR/cross$1.cfg

setenv MERGE_INDEX $TRANSIMS_HOME/bin/MergeIndices
setenv INDEXDEFRAG $TRANSIMS_HOME/bin/IndexDefrag
setenv COLLATOR $TRANSIMS_HOME/bin/Collator
setenv ACTREGEN $TRANSIMS_HOME/bin/ActivityRegenerator.tt3
setenv HOUSEHOLDS $TRANSIMS_HOME/bin/MakeHouseholdFile
setenv DTOA $TRANSIMS_HOME/bin/10to26
setenv INDEX_ACT $TRANSIMS_HOME/bin/IndexActivityFile
setenv PERL /usr/bin/perl
setenv GAWK /usr/bin/gawk

set mxavail = `tail -100 $ACT_DIR/machines.$1`
@ mstart = 0
set length = `wc -l $ACT_DIR/machines.$1`
@ mend = $length[1]
@ muse = $5
@ c = $3
@ cc = $2

# check for continuation run
if ($6 > 1) then
    echo "Continuation of previous iteration" >>&! $LOG_FILE
```

```

echo "ACTIVITY_FILE" $ACTIVITY_FILE >> $TMP_CONFIG
goto loop
endif

# Regenerate the activities that crossed the river
echo "Regenerating activities to remove river crossings..." >>&! $LOG_FILE

/bin/cp $CONFIG_FILE $TMP_CONFIG
/bin/chmod u+w $TMP_CONFIG

echo "ROUTER_HOUSEHOLD_FILE" $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "SEL_HOUSEHOLD_FILE" $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "ACTIVITY_FILE" $ACTIVITY_FILE >> $TMP_CONFIG
echo "SEL_ITDB_FILE" $ITDB_FILE >> $TMP_CONFIG
echo "ACT_FEEDBACK_FILE" $FEEDBACK_COMMANDS >> $TMP_CONFIG
echo "ACT_LOG_FILE" $ACT_LOG_FILE >> $TMP_CONFIG

echo "Dividing households among $muse processors" >>&! $LOG_FILE
$HOUSEHOLDS $TMP_CONFIG $muse >>&! $LOG_FILE

echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $ACTIVITY_FILE >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then
        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
            set mxused = ($mxused $mx)
            @ mused++
        endif
        @ m++
    else
        echo "No response from $mx"
    endif
end
# Spawn processes
@ m = 0
echo "Running Collator" >>&! $LOG_FILE
foreach mx ($mxused)
    set suffix = `DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
    @ m++
end
# Wait for processes to finish on all cpus

```

```

set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.000.it >! $ITDB_FILE.000.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.000.it >> $ITDB_FILE.000.it
    @ m++
end

echo "Finding river crossings in iteration database 0 " >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.000.it $FEEDBACK_COMMANDS $LOG_FILE $1 1

loop:
set i = $6
echo "Running Activity Regenerator for iteration" $i "with c =" $c >>&! $LOG_FILE
echo "ACT_PARTIAL_OUTPUT" $PARTIAL_FILE.$i >> $TMP_CONFIG
echo "ACT_PROBLEM_FILE" $PROBLEM_FILE.$i >> $TMP_CONFIG
echo "ACT_TRAVEL_TIME_FUNCTION_PARAMETERS" $cc\;$c >> $TMP_CONFIG
$ACTREGEN $TMP_CONFIG >>&! $LOG_FILE

# remove indices created by ActivityRegenerator because sometimes they are corrupted
/bin/rm -f $PARTIAL_FILE.$i.*.idx
echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $PARTIAL_FILE.$i >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
    set result = `ping -q -c 2 $mx | grep "0 packets received"`
    if ("X$result" == "X") then
        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
            set mxused = ($mxused $mx)
            @ mused++
        endif
        @ m++
    else
        echo "No response from $mx"
    end
end

```

```

        endif
end
echo "ACTIVITY_FILE " $PARTIAL_FILE.$i >> $TMP_CONFIG
sleep 10
echo "Running Collator" >>&! $LOG_FILE
# Spawn processes
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    echo "Collator process $m on $mx with suffix $suffix"
    echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
    ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
    @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
    set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    while ($numJobs > 0)
        sleep 10
        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
    end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
    set suffix = `$DTOA $m`
    if ($m == 0) then
        head -2 $ITDB_FILE.t$suffix.00$i.it >! $ITDB_FILE.00$i.it
    endif
    $GAWK 'NR>2' $ITDB_FILE.t$suffix.00$i.it >> $ITDB_FILE.00$i.it
    @ m++
end

echo "Finding river crossings in iteration database" $i >>&! $LOG_FILE
$PERL $SCRIPT_DIR/FindCrossings.pl $ITDB_FILE.00$i.it crossjunk$1 $LOG_FILE $1 0

# Merge the partial activity set indexes with the original activity set index.
echo "Merging partial activity indexes.. " >>& $LOG_FILE
$MERGE_INDEX $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.hh.idx $PARTIAL_FILE.$i.hh.idx >>&! $LOG_FILE

# Create a merged activity file from the merged indexes
echo "Creating merged activity file.." >>& $LOG_FILE
$INDEXDEFrag $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.$1 >>&! $LOG_FILE

echo "Finished iteration $i." >>& $LOG_FILE

```

3. TRAVEL TIME FUNCTIONS

3.1 TravelTimeFunction.C.NS

Travel Time Function for North/South Bridge Crossings

```
#ifdef CASESTUDY

// Portland travel time function based on river crossings
// and zone to zone travel times from file.
#include <vector>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <ACT/ACTError.h>
#include <ACT/TravelTimeFunction.h>
#include <ACT/Zone.h>
#include <IO/actio.h>
#include <GBL/TLog.h>
#include <GBL/Log.h>
#include <GBL/Random.h>

int region1[] = {999, 970, 982, 1232, 1225, 1226, 1221, 1231, 1234, 997, 998,
    994, 991, 1059, 1058, 995, 1054, 1053, 993, 992, 983, 988, 989, 1051,
    1052, 990, 1050, 984, 1043, 985, 986, 987, 1038, 1039, 1040, 1042,
    1041, 978, 977, 979, 980, 981, 1017, 1018, 1027, 1029, 1028, 1021,
    1022, 1019, 1020, 973, 1005, 1006, 972, 1012, 971, 1007, 1003, 1008,
    1004, 1000, 974, 975, 976, 1049, 1063, 1048, 1047, 1046, 1057, 1037,
    1036, 1044, 1035, 1045, 1031, 1032, 1033, 1030, 1023, 1024, 1025,
    1013, 1014, 1015, 1009, 1010, 1001, 1062, 1149, 1139, 1235, 1233,
    1220, 1117, 1116, 1118, 1115, 1119, 1194, 1197, 1203, 1187, 1188,
    1198, 1204, 1201, 1205, 1196, 1207, 1206, 1195, 1208, 1193, 1209,
    1211, 1210, 996, 1072, 1065, 1060, 1055, 1064, 1034, 1136, 1011,
    1016, 1002, 1185, 1244, 1241, 1243, 1242, 1240, 1236, 1227, 1230,
    1229, 1114, 1228, 1237, 1224, 1110, 1113, 1111, 1222, 1223, 1112,
    1174, 1219, 1108, 1109, 1213, 1214, 1215, 1216, 1217, 1218, 1173,
    1172, 1212, 1171, 1107, 1167, 1102, 1168, 1105, 1170, 1169, 1239,
    1238, 1177, 1176, 1175, 1106, 1103, 1084, 1104, 1096, 1097, 1098,
```

```
1082, 1166, 1101, 1083, 1081, 1100, 1080, 1099, 1092, 1093, 1094,
1095, 1078, 1079, 1075, 1089, 1076, 1077, 1090, 1069, 1091, 1086,
1087, 1088, 1073, 1074, 1066, 1070, 1085, 1071, 1061, 1056, 1162,
1163, 1164, 1165, 1160, 1161, 1159, 1178, 1158, 1157, 1067, 1068,
1152, 1153, 1154, 1155, 1156, 1147, 1148, 1179, 1150, 1151, 1146,
1144, 1143, 1145, 1140, 1141, 1142, 1183, 1137, 1138, 1133, 1134,
1135, 1184, 1026, 1129, 1130, 1132, 1125, 1131, 1126, 1127, 1128,
1186, 1121, 1199, 1122, 1123, 1124, 1120, 1200, 1202, 1191, 1190,
1192, 1182, 1180, 1189, 1181, 1245, 1246};

int region2[] = {270, 271, 272, 273, 268, 274, 269, 275, 276, 206, 277, 236,
267, 207, 167, 279, 278, 208, 205, 166, 129, 168, 204, 280, 223,
281, 169, 237, 234, 235, 259, 258, 225, 224, 282, 103, 130, 171,
24, 209, 165, 210, 170, 164, 262, 203, 131, 257, 233, 212, 226,
211, 213, 286, 260, 172, 227, 232, 104, 163, 202, 261, 256, 238,
201, 102, 197, 162, 132, 266, 198, 264, 265, 239, 263, 215, 105,
173, 255, 231, 128, 240, 101, 200, 199, 214, 194, 174, 133, 195,
161, 283, 284, 251, 252, 253, 285, 250, 254, 192, 196, 228, 249,
241, 193, 242, 216, 79, 175, 106, 230, 127, 191, 229, 136, 160,
108, 80, 190, 100, 218, 217, 107, 134, 243, 301, 137, 99, 159, 81,
82, 135, 244, 110, 189, 188, 176, 109, 177, 187, 299, 138, 126, 111,
245, 98, 219, 39, 139, 186, 112, 185, 184, 246, 140, 178, 141, 158,
157, 125, 83, 155, 97, 96, 183, 302, 300, 142, 303, 179, 156, 124,
144, 113, 145, 143, 84, 154, 220, 85, 248, 182, 123, 87, 86, 180,
146, 95, 122, 247, 153, 114, 287, 94, 298, 181, 91, 121, 152, 147,
304, 305, 88, 115, 306, 120, 148, 297, 118, 93, 116, 221, 90, 151,
92, 117, 89, 222, 316, 149, 119, 308, 310, 317, 309, 295, 150, 339,
315, 340, 296, 933, 318, 314, 319, 311, 288, 313, 932, 320, 307, 338,
402, 403, 312, 336, 321, 337, 341, 294, 322, 335, 332, 344, 342, 289,
394, 334, 333, 343, 345, 348, 346, 349, 350, 347, 365, 366, 293, 362,
363, 364, 291, 367, 290, 373, 368, 371, 372, 370, 292, 369, 374, 396,
397, 392, 391, 395, 399, 384, 393, 398, 383, 67, 361,
930, 26, 27, 25, 19, 23, 20, 22, 17, 18, 21, 8, 7, 28, 29, 5,
30, 6, 9, 4, 3, 962, 10, 1, 2, 31, 11, 12, 34, 13, 32, 16, 33, 15,
14, 36, 44, 35, 38, 43, 42, 37, 47, 48, 46, 948, 40, 51, 41, 50, 947,
52, 946, 945, 53, 57, 56, 49, 58, 54, 55, 59, 61, 64, 60, 62, 63,
944, 65, 66, 70, 71, 69, 72, 73, 75, 68, 74, 77, 76, 78, 325, 326,
327, 324, 323, 328, 329, 331, 330, 353, 355, 354, 352, 351, 356, 360,
357, 359, 375, 358, 377, 378, 376, 379, 380, 381, 382, 385, 400, 390,
931, 520, 386, 389, 387, 388, 401, 45, 1247, 1248, 1249, 1250, 1251, 1252,
1253};

int region3[] = {925, 926, 911, 961, 909, 960, 924, 923, 910, 908, 912, 904,
963, 959, 921, 922, 918, 917, 920, 915, 907, 919, 913, 906, 916, 958,
914, 905, 902, 903, 852, 968, 885, 886, 884, 880, 881, 957, 956, 882,
883, 964, 969, 887, 879, 955, 966, 878, 954, 877, 876, 875, 874, 873,
872, 888, 967, 965, 892, 864, 865, 953, 952, 867, 866, 870, 868, 871,
869, 889, 851, 951, 863, 862, 861, 860, 857, 859, 890, 858, 891, 929,
716, 853, 854, 855, 928, 722, 720, 856, 950, 848, 714, 715, 721, 717,
718, 771, 847, 849, 767, 766, 846, 719, 768, 769, 772, 770, 765, 764,
763, 774, 727, 773, 781, 783, 784, 786, 779, 777, 782, 949, 785, 787,
780, 778, 776, 775, 788, 790, 791, 798, 799, 800, 789, 793, 807, 808,
815, 816, 792, 802, 797, 801, 806, 814, 817, 809, 794, 795, 813, 804,
```

796, 805, 803, 810, 818, 811, 830, 831, 827, 826, 823, 812, 822, 819,
 828, 825, 824, 821, 820, 833, 832, 829, 845, 838, 844, 839, 841, 842,
 843, 834, 835, 836, 837, 468, 840, 418, 556, 555, 927, 1292, 678, 633,
 679, 677, 634, 631, 702, 703, 637, 680, 676, 635, 675, 681, 701, 700,
 636, 682, 647, 648, 699, 698, 704, 638, 646, 649, 650, 674, 639, 683,
 673, 651, 697, 645, 671, 672, 684, 695, 696, 644, 652, 705, 643, 708,
 670, 640, 642, 694, 654, 685, 669, 655, 653, 668, 656, 657, 658, 667,
 641, 605, 686, 660, 659, 666, 709, 585, 687, 693, 661, 665, 664, 584,
 663, 688, 692, 706, 662, 689, 707, 691, 690, 542, 583, 550, 901, 900,
 894, 899, 895, 893, 898, 607, 723, 624, 606, 896, 617, 897, 618, 623,
 616, 608, 622, 619, 724, 609, 625, 615, 620, 621, 610, 626, 614, 630,
 725, 627, 611, 612, 613, 628, 629, 747, 746, 726, 748, 762, 761, 599,
 597, 598, 745, 596, 749, 573, 744, 760, 600, 595, 728, 743, 742, 759,
 572, 593, 601, 750, 594, 602, 751, 752, 758, 713, 571, 740, 741, 753,
 591, 592, 729, 603, 754, 757, 588, 590, 730, 574, 755, 739, 570, 850,
 756, 575, 569, 568, 567, 587, 589, 604, 566, 563, 737, 576, 738, 586,
 577, 731, 732, 736, 562, 578, 579, 733, 581, 564, 735, 561, 580, 734,
 558, 559, 560, 711, 565, 712, 710, 582, 632, 1290, 470, 476, 539, 541,
 466, 467, 540, 469, 551, 474, 406, 940, 557, 408, 413, 407, 422, 440,
 442, 436, 410, 411, 412, 465, 471, 941, 443, 939, 423, 943, 942, 409,
 424, 477, 425, 435, 475, 426, 545, 428, 549, 464, 444, 427, 432, 445,
 472, 544, 543, 548, 433, 473, 463, 434, 478, 480, 552, 481, 447, 553,
 429, 479, 430, 446, 461, 431, 486, 448, 485, 458, 459, 449, 488, 487,
 547, 484, 451, 452, 489, 450, 546, 482, 490, 462, 491, 483, 404, 460,
 497, 454, 498, 1291, 453, 554, 457, 496, 503, 938, 502, 495, 494, 538,
 456, 455, 504, 534, 493, 492, 501, 937, 500, 505, 499, 506, 934, 936,
 509, 510, 935, 529, 508, 511, 514, 515, 507, 530, 513, 533, 516, 512,
 531, 517, 526, 527, 518, 523, 519, 536, 537, 524, 522, 528, 521,
 532, 525, 535, 416, 415, 405, 439, 441, 419, 438, 420, 417, 414, 437,
 421, 1254, 1255, 1256, 1257, 1258, 1259, 1260};

```
char ttFtravel_time_file[1024];

/** Travel Times by zone pairs for Portland. ***/
typedef struct times_s {
    REAL times[1293][1293];
} TIMES;

// NORTH TO SOUTH Penalty
// Travel time function for Portland. param[0] is a constant
// to be added to the travel time for trips that cross bridges
// north to south.
// Uses travel time file to determine zone to zone travel times
// for car, bike, transit, light rail, park and ride, and walk modes.
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time, TZone& ZoneFrom, TZone& ZoneTo, REAL
    fromX, REAL fromY, REAL toX, REAL toY, TLog& logstream, TRandom& randomstream, vector<double>& params)
{
```

```

static map<int, short> region;
static data_loaded = 0;
// Zone to zone travel times by time intervals.
static TIMES auto_tt[5];           // Mode 2
static TIMES transit_tt[5]; // Mode 3
static TIMES walk_tt[1];           // Mode 1
static TIMES bike_tt[1];           // Mode 7
static TIMES lrail_tt[1];          // Mode 4
static TIMES prout_tt[1];          // Mode 5
static TIMES prin_tt[1];           // Mode 6
static const int North = 1;
static const int West = 2;
static const int East = 3;
int index = 0;
int params_okay = 0;

// Need a travel time constant in the parameters array to
// penalize the bridge crossings.
if (params.size() == 1)
    params_okay = 1;

if (!data_loaded) {
    memset(auto_tt, 0, 5*sizeof(TIMES));
    memset(transit_tt, 0, 5*sizeof(TIMES));
    memset(walk_tt, 0, sizeof(TIMES));
    memset(bike_tt, 0, sizeof(TIMES));
    memset(lrail_tt, 0, sizeof(TIMES));
    memset(prout_tt, 0, sizeof(TIMES));
    memset(prin_tt, 0, sizeof(TIMES));
    logstream.print(LOG_ACT) << "TTravelTimeFunction: Loading travel times from file" << done;
    FILE *fp = fopen(ttFtravel_time_file, "r");
    if (!fp) {
        logstream.fatal(SUB_ACT, TERR_ACT_UNREADABLE_FILE)
        << "TTravelTimeFunction: Failed to open zone to zone travel times file "
        << ttFtravel_time_file << done;
    }
    while (moreTravelTimes(fp)) {
        const TTravelTimeEntry *t = getTravelTimeEntryFromFile(fp);
        if (!t)
            continue;
        switch (t->fMode) {
            case 1:
                // Walk - one time interval for walk mode.
                walk_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
                break;
            case 2:
                // Car - 5 time intervals for car mode (1 - 5).
                // Array indexes 0 - 4.
        }
    }
}

```

```

        if (t->fInterval < 6)
            auto_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
        break;
    case 3:
        // Transit - 5 time intervals for transit (1 - 5).
        // Array indexes 0 - 4.
        if (t->fInterval < 6)
            transit_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
        break;
    case 4:
        // MAX - 1 time interval for MAX.
        lrail_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    case 5:
        // Park and Ride out - 1 time interval for P&R out.
        prout_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    case 6:
        // Park and Ride in - 1 time interval for P&R in.
        prin_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;
    case 7:
        // Bicycle - 1 time interval for bicycle.
        bike_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    default:
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Unexpected mode "
        << t->fMode << " for zones/interval " << t->fZone1
        << "/" << t->fZone2 << "/" << t->fInterval << " ignored." << done;
        break;
    }
}
data_loaded = 1;
logstream.print(LOG_ACT) << "TTravelTimeFunction: Finished loading travel times from file" << done;
fclose(fp);
}

if (region.empty()) {
    for (int i = 0; i < sizeof(region1) / sizeof(int); ++i)
        region[region1[i]] = 1;
    for (int i = 0; i < sizeof(region2) / sizeof(int); ++i)
        region[region2[i]] = 2;
    for (int i = 0; i < sizeof(region3) / sizeof(int); ++i)
        region[region3[i]] = 3;
}

```

```

}

if (ZoneFrom.getTTFunctionParam() == -1) {
    if (!region.count(ZoneFrom.getNumber())) {
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Zone " << ZoneFrom.getNumber()
        << " missing from region list." << done;
        return 0.0;
    }
    ZoneFrom.setTTFunctionParam(region[ZoneFrom.getNumber()]);
}

if (ZoneTo.getTTFunctionParam() == -1) {
    if (!region.count(ZoneTo.getNumber())) {
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Zone " << ZoneTo.getNumber()
        << " missing from region list." << done;
        return 0.0;
    }
    ZoneTo.setTTFunctionParam(region[ZoneTo.getNumber()]);
}

REAL the_time = 0.0;
switch (mode) {
    case 1:
        // Walk
        the_time = walk_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 2:
        // Car
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)
            index = 4;
        the_time = auto_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 3:
        // Transit index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)

```

```

        index = 2;
    else if (time >= 16.5 && time < 18.5)
        index = 3;
    else if (time >= 18.5)
        index = 4;
    the_time = transit_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
    break;
case 4:
    // MAX
    the_time = lrail_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
    break;

case 5:
    // Park and Ride Out
    the_time = prout_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
    break;

case 6:
    // Park and Ride In
    the_time = prin_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
    break;
case 7:
    // Bike
    the_time = bike_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
    break;

default:
    break;
}

// Penalize travel time between regions.
if (params_okay) {
    int destSouth = 0;
    if ((ZoneTo.getTTFunctionParam() == West || ZoneTo.getTTFunctionParam() == East))
        destSouth = 1;
    // Penalize crossing from north to south with first parameter.
    if ((ZoneFrom.getTTFunctionParam() == North) && destSouth)
        the_time += params[0];
}

// Do not return a negative travel time.
if (the_time < 0.0)
    return 0.0;

return the_time;

```

```
}
```

```
#else
```

```
// Generic travel time function...doesn't do anything.
```

```
#include <math.h>
```

```
#include <ACT/ACTError.h>
#include <ACT/TravelTimeFunction.h>
#include <ACT/Zone.h>
#include <GBL/TLog.h>
#include <GBL/Random.h>
```

```
// Get the travel time for the given mode using the function
// supplied in the method. Returns travel time in seconds.
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time,
    TZone& ZoneFrom, TZone& ZoneTo, REAL fromX, REAL fromY, REAL toX, REAL toY,
    TLog& logstream, TRandom& randomstream, vector<double>& params)
{
    return 0.0;
}
#endif
```

3.2 TravelTimeFunction.C.SN

Travel Time Function for South/North Bridge Crossings

```
#ifdef CASESTUDY

// Portland travel time function based on river crossings
// and zone to zone travel times from file.
#include <vector>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <ACT/ACTError.h>
#include <ACT/TravelTimeFunction.h>
#include <ACT/Zone.h>
#include <IO/actio.h>
#include <GBL/TLog.h>
#include <GBL/Log.h>
#include <GBL/Random.h>

int region1[] = {999, 970, 982, 1232, 1225, 1226, 1221, 1231, 1234, 997, 998,
 994, 991, 1059, 1058, 995, 1054, 1053, 993, 992, 983, 988, 989, 1051,
 1052, 990, 1050, 984, 1043, 985, 986, 987, 1038, 1039, 1040, 1042,
 1041, 978, 977, 979, 980, 981, 1017, 1018, 1027, 1029, 1028, 1021,
 1022, 1019, 1020, 973, 1005, 1006, 972, 1012, 971, 1007, 1003, 1008,
 1004, 1000, 974, 975, 976, 1049, 1063, 1048, 1047, 1046, 1057, 1037,
 1036, 1044, 1035, 1045, 1031, 1032, 1033, 1030, 1023, 1024, 1025,
 1013, 1014, 1015, 1009, 1010, 1001, 1062, 1149, 1139, 1235, 1233,
 1220, 1117, 1116, 1118, 1115, 1119, 1194, 1197, 1203, 1187, 1188,
 1198, 1204, 1201, 1205, 1196, 1207, 1206, 1195, 1208, 1193, 1209,
 1211, 1210, 996, 1072, 1065, 1060, 1055, 1064, 1034, 1136, 1011,
 1016, 1002, 1185, 1244, 1241, 1243, 1242, 1240, 1236, 1227, 1230,
 1229, 1114, 1228, 1237, 1224, 1110, 1113, 1111, 1222, 1223, 1112,
 1174, 1219, 1108, 1109, 1213, 1214, 1215, 1216, 1217, 1218, 1173,
 1172, 1212, 1171, 1107, 1167, 1102, 1168, 1105, 1170, 1169, 1239,
 1238, 1177, 1176, 1175, 1106, 1103, 1084, 1104, 1096, 1097, 1098,
 1082, 1166, 1101, 1083, 1081, 1100, 1080, 1099, 1092, 1093, 1094,
 1095, 1078, 1079, 1075, 1089, 1076, 1077, 1090, 1069, 1091, 1086,
 1087, 1088, 1073, 1074, 1066, 1070, 1085, 1071, 1061, 1056, 1162,
 1163, 1164, 1165, 1160, 1161, 1159, 1178, 1158, 1157, 1067, 1068,
```

```
1152, 1153, 1154, 1155, 1156, 1147, 1148, 1179, 1150, 1151, 1146,
1144, 1143, 1145, 1140, 1141, 1142, 1183, 1137, 1138, 1133, 1134,
1135, 1184, 1026, 1129, 1130, 1132, 1125, 1131, 1126, 1127, 1128,
1186, 1121, 1199, 1122, 1123, 1124, 1120, 1200, 1191, 1190,
1192, 1182, 1180, 1189, 1181, 1245, 1246};

int region2[] = {270, 271, 272, 273, 268, 274, 269, 275, 276, 206, 277, 236,
267, 207, 167, 279, 278, 208, 205, 166, 129, 168, 204, 280, 223,
281, 169, 237, 234, 235, 259, 258, 225, 224, 282, 103, 130, 171,
24, 209, 165, 210, 170, 164, 262, 203, 131, 257, 233, 212, 226,
211, 213, 286, 260, 172, 227, 232, 104, 163, 202, 261, 256, 238,
201, 102, 197, 162, 132, 266, 198, 264, 265, 239, 263, 215, 105,
173, 255, 231, 128, 240, 101, 200, 199, 214, 194, 174, 133, 195,
161, 283, 284, 251, 252, 253, 285, 250, 254, 192, 196, 228, 249,
241, 193, 242, 216, 79, 175, 106, 230, 127, 191, 229, 136, 160,
108, 80, 190, 100, 218, 217, 107, 134, 243, 301, 137, 99, 159, 81,
82, 135, 244, 110, 189, 188, 176, 109, 177, 187, 299, 138, 126, 111,
245, 98, 219, 39, 139, 186, 112, 185, 184, 246, 140, 178, 141, 158,
157, 125, 83, 155, 97, 96, 183, 302, 300, 142, 303, 179, 156, 124,
144, 113, 145, 143, 84, 154, 220, 85, 248, 182, 123, 87, 86, 180,
146, 95, 122, 247, 153, 114, 287, 94, 298, 181, 91, 121, 152, 147,
304, 305, 88, 115, 306, 120, 148, 297, 118, 93, 116, 221, 90, 151,
92, 117, 89, 222, 316, 149, 119, 308, 310, 317, 309, 295, 150, 339,
315, 340, 296, 933, 318, 314, 319, 311, 288, 313, 932, 320, 307, 338,
402, 403, 312, 336, 321, 337, 341, 294, 322, 335, 332, 344, 342, 289,
394, 334, 333, 343, 345, 348, 346, 349, 350, 347, 365, 366, 293, 362,
363, 364, 291, 367, 290, 373, 368, 371, 372, 370, 292, 369, 374, 396,
397, 392, 391, 395, 399, 384, 393, 398, 383, 67, 361,
930, 26, 27, 25, 19, 23, 20, 22, 17, 18, 21, 8, 7, 28, 29, 5,
30, 6, 9, 4, 3, 962, 10, 1, 2, 31, 11, 12, 34, 13, 32, 16, 33, 15,
14, 36, 44, 35, 38, 43, 42, 37, 47, 48, 46, 948, 40, 51, 41, 50, 947,
52, 946, 945, 53, 57, 56, 49, 58, 54, 55, 59, 61, 64, 60, 62, 63,
944, 65, 66, 70, 71, 69, 72, 73, 75, 68, 74, 77, 76, 78, 325, 326,
327, 324, 323, 328, 329, 331, 330, 353, 355, 354, 352, 351, 356, 360,
357, 359, 375, 358, 377, 378, 376, 379, 380, 381, 382, 385, 400, 390,
931, 520, 386, 389, 387, 388, 401, 45, 1247, 1248, 1249, 1250, 1251, 1252,
1253};

int region3[] = {925, 926, 911, 961, 909, 960, 924, 923, 910, 908, 912, 904,
963, 959, 921, 922, 918, 917, 920, 915, 907, 919, 913, 906, 916, 958,
914, 905, 902, 903, 852, 968, 885, 886, 884, 880, 881, 957, 956, 882,
883, 964, 969, 887, 879, 955, 966, 878, 954, 877, 876, 875, 874, 873,
872, 888, 967, 965, 892, 864, 865, 953, 952, 867, 866, 870, 868, 871,
869, 889, 851, 951, 863, 862, 861, 860, 857, 859, 890, 858, 891, 929,
716, 853, 854, 855, 928, 722, 720, 856, 950, 848, 714, 715, 721, 717,
718, 771, 847, 849, 767, 766, 846, 719, 768, 769, 772, 770, 765, 764,
763, 774, 727, 773, 781, 783, 784, 786, 779, 777, 782, 949, 785, 787,
780, 778, 776, 775, 788, 790, 791, 798, 799, 800, 789, 793, 807, 808,
815, 816, 792, 802, 797, 801, 806, 814, 817, 809, 794, 795, 813, 804,
796, 805, 803, 810, 818, 811, 830, 831, 827, 826, 823, 812, 822, 819,
828, 825, 824, 821, 820, 833, 832, 829, 845, 838, 844, 839, 841, 842,
843, 834, 835, 836, 837, 468, 840, 418, 556, 555, 927, 1292, 678, 633,
679, 677, 634, 631, 702, 703, 637, 680, 676, 635, 675, 681, 701, 700,
```

636, 682, 647, 648, 699, 698, 704, 638, 646, 649, 650, 674, 639, 683,
 673, 651, 697, 645, 671, 672, 684, 695, 696, 644, 652, 705, 643, 708,
 670, 640, 642, 694, 654, 685, 669, 655, 653, 668, 656, 657, 658, 667,
 641, 605, 686, 660, 659, 666, 709, 585, 687, 693, 661, 665, 664, 584,
 663, 688, 692, 706, 662, 689, 707, 691, 690, 542, 583, 550, 901, 900,
 894, 899, 895, 893, 898, 607, 723, 624, 606, 896, 617, 897, 618, 623,
 616, 608, 622, 619, 724, 609, 625, 615, 620, 621, 610, 626, 614, 630,
 725, 627, 611, 612, 613, 628, 629, 747, 746, 726, 748, 762, 761, 599,
 597, 598, 745, 596, 749, 573, 744, 760, 600, 595, 728, 743, 742, 759,
 572, 593, 601, 750, 594, 602, 751, 752, 758, 713, 571, 740, 741, 753,
 591, 592, 729, 603, 754, 757, 588, 590, 730, 574, 755, 739, 570, 850,
 756, 575, 569, 568, 567, 587, 589, 604, 566, 563, 737, 576, 738, 586,
 577, 731, 732, 736, 562, 578, 579, 733, 581, 564, 735, 561, 580, 734,
 558, 559, 560, 711, 565, 712, 710, 582, 632, 1290, 470, 476, 539, 541,
 466, 467, 540, 469, 551, 474, 406, 940, 557, 408, 413, 407, 422, 440,
 442, 436, 410, 411, 412, 465, 471, 941, 443, 939, 423, 943, 942, 409,
 424, 477, 425, 435, 475, 426, 545, 428, 549, 464, 444, 427, 432, 445,
 472, 544, 543, 548, 433, 473, 463, 434, 478, 480, 552, 481, 447, 553,
 429, 479, 430, 446, 461, 431, 486, 448, 485, 458, 459, 449, 488, 487,
 547, 484, 451, 452, 489, 450, 546, 482, 490, 462, 491, 483, 404, 460,
 497, 454, 498, 1291, 453, 554, 457, 496, 503, 938, 502, 495, 494, 538,
 456, 455, 504, 534, 493, 492, 501, 937, 500, 505, 499, 506, 934, 936,
 509, 510, 935, 529, 508, 511, 514, 515, 507, 530, 513, 533, 516, 512,
 531, 517, 526, 527, 518, 523, 519, 536, 537, 524, 522, 528, 521,
 532, 525, 535, 416, 415, 405, 439, 441, 419, 438, 420, 417, 414, 437,
 421, 1254, 1255, 1256, 1257, 1258, 1259, 1260};

```
char ttFtravel_time_file[1024];

/** Travel Times by zone pairs for Portland. ***/
typedef struct times_s {
    REAL times[1293][1293];
} TIMES;

// SOUTH TO NORTH Penalty
// Travel time function for Portland. param[0] is a constant
// to be added to the travel time for trips that cross bridges
// south to north.
// Uses travel time file to determine zone to zone travel times
// for car, bike, transit, light rail, park and ride, and walk modes.
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time, TZone& ZoneFrom, TZone& ZoneTo, REAL
    fromX, REAL fromY, REAL toX, REAL toY, TLog& logstream, TRandom& randomstream, vector<double>& params)
{

    static map<int, short> region;
    static data_loaded = 0;
    // Zone to zone travel times by time intervals.
```

```

static TIMES auto_tt[5];           // Mode 2
static TIMES transit_tt[5]; // Mode 3
static TIMES walk_tt[1];          // Mode 1
static TIMES bike_tt[1];          // Mode 7
static TIMES lrail_tt[1];         // Mode 4
static TIMES prout_tt[1];         // Mode 5
static TIMES prin_tt[1];          // Mode 6
static const int North = 1;
static const int West = 2;
static const int East = 3;
int index = 0;
int params_okay = 0;

// Need a travel time constant in the parameters array to
// penalize the bridge crossings.
if (params.size() == 1)
    params_okay = 1;

if (!data_loaded) {
    memset(auto_tt, 0, 5*sizeof(TIMES));
    memset(transit_tt, 0, 5*sizeof(TIMES));
    memset(walk_tt, 0, sizeof(TIMES));
    memset(bike_tt, 0, sizeof(TIMES));
    memset(lrail_tt, 0, sizeof(TIMES));
    memset(prout_tt, 0, sizeof(TIMES));
    memset(prin_tt, 0, sizeof(TIMES));
    logstream.print(LOG_ACT) << "TTravelTimeFunction: Loading travel times from file" << done;
    FILE *fp = fopen(ttFtravel_time_file, "r");
    if (!fp) {
        logstream.fatal(SUB_ACT, TERR_ACT_UNREADABLE_FILE)
        << "TTravelTimeFunction: Failed to open zone to zone travel times file "
        << ttFtravel_time_file << done;
    }
    while (moreTravelTimes(fp)) {
        const TTravelTimeEntry *t = getTravelTimeEntryFromFile(fp);
        if (!t)
            continue;
        switch (t->fMode) {
        case 1:
            // Walk - one time interval for walk mode.
            walk_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
            break;
        case 2:
            // Car - 5 time intervals for car mode (1 - 5).
            // Array indexes 0 - 4.
            if (t->fInterval < 6)
                auto_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
            break;
        case 3:
    }
}

```

```

        // Transit - 5 time intervals for transit (1 - 5).
        // Array indexes 0 - 4.
        if (t->fInterval < 6)
            transit_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
        break;
    case 4:
        // MAX - 1 time interval for MAX.
        ltrail_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    case 5:
        // Park and Ride out - 1 time interval for P&R out.
        prout_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    case 6:
        // Park and Ride in - 1 time interval for P&R in.
        prin_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;
    case 7:
        // Bicycle - 1 time interval for bicycle.
        bike_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    default:
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Unexpected mode "
        << t->fMode << " for zones/interval " << t->fZone1
        << "/" << t->fZone2 << "/" << t->fInterval << " ignored." << done;
        break;
    }
}
data_loaded = 1;
logstream.print(LOG_ACT) << "TTravelTimeFunction: Finished loading travel times from file" << done;
fclose(fp);
}

if (region.empty()) {
    for (int i = 0; i < sizeof(region1) / sizeof(int); ++i)
        region[region1[i]] = 1;
    for (int i = 0; i < sizeof(region2) / sizeof(int); ++i)
        region[region2[i]] = 2;
    for (int i = 0; i < sizeof(region3) / sizeof(int); ++i)
        region[region3[i]] = 3;
}

if (ZoneFrom.getTTFunctionParam() == -1) {

```

```

if (!region.count(ZoneFrom.getNumber())) {
    logstream.warning(SUB_ACT) << "TTtravelTimeFunction: Zone " << ZoneFrom.getNumber()
    << " missing from region list." << done;
    return 0.0;
}
ZoneFrom.setTTFunctionParam(region[ZoneFrom.getNumber()]);
}

if (ZoneTo.getTTFunctionParam() == -1) {
if (!region.count(ZoneTo.getNumber())) {
    logstream.warning(SUB_ACT) << "TTtravelTimeFunction: Zone " << ZoneTo.getNumber()
    << " missing from region list." << done;
    return 0.0;
}
ZoneTo.setTTFunctionParam(region[ZoneTo.getNumber()]);
}

REAL the_time = 0.0;
switch (mode) {
    case 1:
        // Walk
        the_time = walk_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 2:
        // Car
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)
            index = 4;
        the_time = auto_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 3:
        // Transit
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)

```

```

        index = 4;
        the_time = transit_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 4:
        // MAX
        the_time = lrail_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;

    case 5:
        // Park and Ride Out
        the_time = prout_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;

    case 6:
        // Park and Ride In
        the_time = prin_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 7:
        // Bike
        the_time = bike_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;

    default:
        break;
}

// Penalize travel time between regions.
if (params_okay) {
    int originSouth = 0;
    if ((ZoneFrom.getTTFParam() == West || ZoneFrom.getTTFParam() == East))
        originSouth = 1;
    // Penalize crossing from south to north with first parameter.
    if ((ZoneTo.getTTFParam() == North) && originSouth)
        the_time += params[0];
}

// Do not return a negative travel time.
if (the_time < 0.0)
    return 0.0;

return the_time;
}

#else

```

```
// Generic travel time function...doesn't do anything.
#include <math.h>

#include <ACT/ACTError.h>
#include <ACT/TravelTimeFunction.h>
#include <ACT/Zone.h>
#include <GBL/TLog.h>
#include <GBL/Random.h>

// Get the travel time for the given mode using the function
// supplied in the method. Returns travel time in seconds.
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time,
    TZone& ZoneFrom, TZone& ZoneTo, REAL fromX, REAL fromY, REAL toX, REAL toY,
    TLog& logstream, TRandom& randomstream, vector<double>& params)
{
    return 0.0;
}
#endif
```

3.3 TravelTimeFunction.C.SN_EW

Travel Time Function for South/North and East/West Bridge Crossings

```
#ifdef CASESTUDY

// Portland travel time function based on river crossings
// and zone to zone travel times from file.
#include <vector>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <ACT/ACTError.h>
#include <ACT/TravelTimeFunction.h>
#include <ACT/Zone.h>
#include <IO/actio.h>
#include <GBL/TLog.h>
#include <GBL/Log.h>
#include <GBL/Random.h>

int region1[] = {999, 970, 982, 1232, 1225, 1226, 1221, 1231, 1234, 997, 998,
 994, 991, 1059, 1058, 995, 1054, 1053, 993, 992, 983, 988, 989, 1051,
 1052, 990, 1050, 984, 1043, 985, 986, 987, 1038, 1039, 1040, 1042,
 1041, 978, 977, 979, 980, 981, 1017, 1018, 1027, 1029, 1028, 1021,
 1022, 1019, 1020, 973, 1005, 1006, 972, 1012, 971, 1007, 1003, 1008,
 1004, 1000, 974, 975, 976, 1049, 1063, 1048, 1047, 1046, 1057, 1037,
 1036, 1044, 1035, 1045, 1031, 1032, 1033, 1030, 1023, 1024, 1025,
 1013, 1014, 1015, 1009, 1010, 1001, 1062, 1149, 1139, 1235, 1233,
 1220, 1117, 1116, 1118, 1115, 1119, 1194, 1197, 1203, 1187, 1188,
 1198, 1204, 1201, 1205, 1196, 1207, 1206, 1195, 1208, 1193, 1209,
 1211, 1210, 996, 1072, 1065, 1060, 1055, 1064, 1034, 1136, 1011,
 1016, 1002, 1185, 1244, 1241, 1243, 1242, 1240, 1236, 1227, 1230,
 1229, 1114, 1228, 1237, 1224, 1110, 1113, 1111, 1222, 1223, 1112,
 1174, 1219, 1108, 1109, 1213, 1214, 1215, 1216, 1217, 1218, 1173,
 1172, 1212, 1171, 1107, 1167, 1102, 1168, 1105, 1170, 1169, 1239,
 1238, 1177, 1176, 1175, 1106, 1103, 1084, 1104, 1096, 1097, 1098,
 1082, 1166, 1101, 1083, 1081, 1100, 1080, 1099, 1092, 1093, 1094,
 1095, 1078, 1079, 1075, 1089, 1076, 1077, 1090, 1069, 1091, 1086,
 1087, 1088, 1073, 1074, 1066, 1070, 1085, 1071, 1061, 1056, 1162,
 1163, 1164, 1165, 1160, 1161, 1159, 1178, 1158, 1157, 1067, 1068,
```

```
1152, 1153, 1154, 1155, 1156, 1147, 1148, 1179, 1150, 1151, 1146,
1144, 1143, 1145, 1140, 1141, 1142, 1183, 1137, 1138, 1133, 1134,
1135, 1184, 1026, 1129, 1130, 1132, 1125, 1131, 1126, 1127, 1128,
1186, 1121, 1199, 1122, 1123, 1124, 1120, 1200, 1191, 1190,
1192, 1182, 1180, 1189, 1181, 1245, 1246};
int region2[] = {270, 271, 272, 273, 268, 274, 269, 275, 276, 206, 277, 236,
267, 207, 167, 279, 278, 208, 205, 166, 129, 168, 204, 280, 223,
281, 169, 237, 234, 235, 259, 258, 225, 224, 282, 103, 130, 171,
24, 209, 165, 210, 170, 164, 262, 203, 131, 257, 233, 212, 226,
211, 213, 286, 260, 172, 227, 232, 104, 163, 202, 261, 256, 238,
201, 102, 197, 162, 132, 266, 198, 264, 265, 239, 263, 215, 105,
173, 255, 231, 128, 240, 101, 200, 199, 214, 194, 174, 133, 195,
161, 283, 284, 251, 252, 253, 285, 250, 254, 192, 196, 228, 249,
241, 193, 242, 216, 79, 175, 106, 230, 127, 191, 229, 136, 160,
108, 80, 190, 100, 218, 217, 107, 134, 243, 301, 137, 99, 159, 81,
82, 135, 244, 110, 189, 188, 176, 109, 177, 187, 299, 138, 126, 111,
245, 98, 219, 39, 139, 186, 112, 185, 184, 246, 140, 178, 141, 158,
157, 125, 83, 155, 97, 96, 183, 302, 300, 142, 303, 179, 156, 124,
144, 113, 145, 143, 84, 154, 220, 85, 248, 182, 123, 87, 86, 180,
146, 95, 122, 247, 153, 114, 287, 94, 298, 181, 91, 121, 152, 147,
304, 305, 88, 115, 306, 120, 148, 297, 118, 93, 116, 221, 90, 151,
92, 117, 89, 222, 316, 149, 119, 308, 310, 317, 309, 295, 150, 339,
315, 340, 296, 933, 318, 314, 319, 311, 288, 313, 932, 320, 307, 338,
402, 403, 312, 336, 321, 337, 341, 294, 322, 335, 332, 344, 342, 289,
394, 334, 333, 343, 345, 348, 346, 349, 350, 347, 365, 366, 293, 362,
363, 364, 291, 367, 290, 373, 368, 371, 372, 370, 292, 369, 374, 396,
397, 392, 391, 395, 399, 384, 393, 398, 383, 67, 361,
930, 26, 27, 25, 19, 23, 20, 22, 17, 18, 21, 8, 7, 28, 29, 5,
30, 6, 9, 4, 3, 962, 10, 1, 2, 31, 11, 12, 34, 13, 32, 16, 33, 15,
14, 36, 44, 35, 38, 43, 42, 37, 47, 48, 46, 46, 948, 40, 51, 41, 50, 947,
52, 946, 945, 53, 57, 56, 49, 58, 54, 55, 59, 61, 64, 60, 62, 63,
944, 65, 66, 70, 71, 69, 72, 73, 75, 68, 74, 77, 76, 78, 325, 326,
327, 324, 323, 328, 329, 331, 330, 353, 355, 354, 352, 351, 356, 360,
357, 359, 375, 358, 377, 378, 376, 379, 380, 381, 382, 385, 400, 390,
931, 520, 386, 389, 387, 388, 401, 45, 1247, 1248, 1249, 1250, 1251, 1252,
1253};
int region3[] = {925, 926, 911, 961, 909, 960, 924, 923, 910, 908, 912, 904,
963, 959, 921, 922, 918, 917, 920, 915, 907, 919, 913, 906, 916, 958,
914, 905, 902, 903, 852, 968, 885, 886, 884, 880, 881, 957, 956, 882,
883, 964, 969, 887, 879, 955, 966, 878, 954, 877, 876, 875, 874, 873,
872, 888, 967, 965, 892, 864, 865, 953, 952, 867, 866, 870, 868, 871,
869, 889, 851, 951, 863, 862, 861, 860, 857, 859, 890, 858, 891, 929,
716, 853, 854, 855, 928, 722, 720, 856, 950, 848, 714, 715, 721, 717,
718, 771, 847, 849, 767, 766, 846, 719, 768, 769, 772, 770, 765, 764,
763, 774, 727, 773, 781, 783, 784, 786, 779, 777, 782, 949, 785, 787,
780, 778, 776, 775, 788, 790, 791, 798, 799, 800, 789, 793, 807, 808,
815, 816, 792, 802, 797, 801, 806, 814, 817, 809, 794, 795, 813, 804,
796, 805, 803, 810, 818, 811, 830, 831, 827, 826, 823, 812, 822, 819,
828, 825, 824, 821, 820, 833, 832, 829, 845, 838, 844, 839, 841, 842,
843, 834, 835, 836, 837, 468, 840, 418, 556, 555, 927, 1292, 678, 633,
679, 677, 634, 631, 702, 703, 637, 680, 676, 635, 675, 681, 701, 700,
```

636, 682, 647, 648, 699, 698, 704, 638, 646, 649, 650, 674, 639, 683,
 673, 651, 697, 645, 671, 672, 684, 695, 696, 644, 652, 705, 643, 708,
 670, 640, 642, 694, 654, 685, 669, 655, 653, 668, 656, 657, 658, 667,
 641, 605, 686, 660, 659, 666, 709, 585, 687, 693, 661, 665, 664, 584,
 663, 688, 692, 706, 662, 689, 707, 691, 690, 542, 583, 550, 901, 900,
 894, 899, 895, 893, 898, 607, 723, 624, 606, 896, 617, 897, 618, 623,
 616, 608, 622, 619, 724, 609, 625, 615, 620, 621, 610, 626, 614, 630,
 725, 627, 611, 612, 613, 628, 629, 747, 746, 726, 748, 762, 761, 599,
 597, 598, 745, 596, 749, 573, 744, 760, 600, 595, 728, 743, 742, 759,
 572, 593, 601, 750, 594, 602, 751, 752, 758, 713, 571, 740, 741, 753,
 591, 592, 729, 603, 754, 757, 588, 590, 730, 574, 755, 739, 570, 850,
 756, 575, 569, 568, 567, 587, 589, 604, 566, 563, 737, 576, 738, 586,
 577, 731, 732, 736, 562, 578, 579, 733, 581, 564, 735, 561, 580, 734,
 558, 559, 560, 711, 565, 712, 710, 582, 632, 1290, 470, 476, 539, 541,
 466, 467, 540, 469, 551, 474, 406, 940, 557, 408, 413, 407, 422, 440,
 442, 436, 410, 411, 412, 465, 471, 941, 443, 939, 423, 943, 942, 409,
 424, 477, 425, 435, 475, 426, 545, 428, 549, 464, 444, 427, 432, 445,
 472, 544, 543, 548, 433, 473, 463, 434, 478, 480, 552, 481, 447, 553,
 429, 479, 430, 446, 461, 431, 486, 448, 485, 458, 459, 449, 488, 487,
 547, 484, 451, 452, 489, 450, 546, 482, 490, 462, 491, 483, 404, 460,
 497, 454, 498, 1291, 453, 554, 457, 496, 503, 938, 502, 495, 494, 538,
 456, 455, 504, 534, 493, 492, 501, 937, 500, 505, 499, 506, 934, 936,
 509, 510, 935, 529, 508, 511, 514, 515, 507, 530, 513, 533, 516, 512,
 531, 517, 526, 527, 518, 523, 519, 536, 537, 524, 522, 528, 521,
 532, 525, 535, 416, 415, 405, 439, 441, 419, 438, 420, 417, 414, 437,
 421, 1254, 1255, 1256, 1257, 1258, 1259, 1260};

```
char ttFtravel_time_file[1024];

/** Travel Times by zone pairs for Portland. ***/
typedef struct times_s {
    REAL times[1293][1293];
} TIMES;

// SOUTH to NORTH and EAST to WEST PENALTY
// Travel time function for Portland. param[0] is a constant
// to be added to the travel time for trips that cross bridges
// (regions) south to north (Regions 2 and 3 into Region 1).
// param[1] is a constant to be added for trips that cross bridges
// (Willamette) (Region 3 to Region 2, east to west).
// Uses travel time file to determine zone to zone travel times
// for car, bike, transit, and walk modes.
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time, TZone& ZoneFrom, TZone& ZoneTo, REAL
                                         fromX, REAL fromY, REAL toX, REAL toY, TLog& logstream, TRandom& randomstream, vector<double>& params)
{
    static map<int, short> region;
```

```

static data_loaded = 0;
// Zone to zone travel times by time intervals.
static TIMES auto_tt[5];           // Mode 2
static TIMES transit_tt[5];         // Mode 3
static TIMES walk_tt[1];            // Mode 1
static TIMES bike_tt[1];            // Mode 7
static const int North = 1;
static const int West = 2;
static const int East = 3;
int index = 0;
int params_okay = 0;

// Need two travel time constants in the parameters arrays to
// penalize the bridge crossings.
if (params.size() == 2)
    params_okay = 1;

if (!data_loaded) {
    memset(auto_tt, 0, 5*sizeof(TIMES));
    memset(transit_tt, 0, 5*sizeof(TIMES));
    memset(walk_tt, 0, sizeof(TIMES));
    memset(bike_tt, 0, sizeof(TIMES));
    logstream.print(LOG_ACT) << "TTravelTimeFunction: Loading travel times from file" << done;
    FILE *fp = fopen(ttFtravel_time_file, "r");
    if (!fp) {
        logstream.fatal(SUB_ACT, TERR_ACT_UNREADABLE_FILE)
        << "TTravelTimeFunction: Failed to open zone to zone travel times file "
        << ttFtravel_time_file << done;
    }
    while (moreTravelTimes(fp)) {
        const TTravelTimeEntry *t = getTravelTimeEntryFromFile(fp);
        if (!t)
            continue;
        switch (t->fMode) {
            case 1:
                // Walk - one time interval for walk mode.
                walk_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
                break;
            case 2:
                // Car - 5 time intervals for car mode (1 - 5).
                // Array indexes 0 - 4.
                if (t->fInterval < 6)
                    auto_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
                break;
            case 3:
                // Transit - 5 time intervals for transit (1 - 5).
                // Array indexes 0 - 4.
                if (t->fInterval < 6)
                    transit_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
        }
    }
}

```

```

        break;
    case 7:
        // Bicycle - 1 time interval for bicycle.
        bike_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    default:
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Unexpected mode "
        << t->fMode << " for zones/interval " << t->fZone1
        << "/" << t->fZone2 << "/" << t->fInterval << " ignored." << done;
        break;
    }
}
data_loaded = 1;
logstream.print(LOG_ACT) << "TTravelTimeFunction: Finished loading travel times from file" << done;
fclose(fp);
}

if (region.empty()) {
    for (int i = 0; i < sizeof(region1) / sizeof(int); ++i)
        region[region1[i]] = 1;
    for (int i = 0; i < sizeof(region2) / sizeof(int); ++i)
        region[region2[i]] = 2;
    for (int i = 0; i < sizeof(region3) / sizeof(int); ++i)
        region[region3[i]] = 3;
}

if (ZoneFrom.getTTFunctionParam() == -1) {
    if (!region.count(ZoneFrom.getNumber())) {
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Zone " << ZoneFrom.getNumber()
        << " missing from region list." << done;
        return 0.0;
    }
    ZoneFrom.setTTFunctionParam(region[ZoneFrom.getNumber()]);
}

if (ZoneTo.getTTFunctionParam() == -1) {
    if (!region.count(ZoneTo.getNumber())) {
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Zone " << ZoneTo.getNumber()
        << " missing from region list." << done;
        return 0.0;
    }
    ZoneTo.setTTFunctionParam(region[ZoneTo.getNumber()]);
}

REAL the_time = 0.0;

```

```

switch (mode) {
    case 1:
        // Walk
        the_time = walk_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 2:
        // Car
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)
            index = 4;
        the_time = auto_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 3:
        // Transit
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)
            index = 4;
        the_time = transit_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 7:
        // Bike
        the_time = bike_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    default:
        break;
}

// Penalize travel time between regions.
if (params_okay) {
    int isSouth = 0;
    if ((ZoneFrom.getTTFUNCTIONParam() == West || ZoneFrom.getTTFUNCTIONParam() == East))
        isSouth = 1;
}

```

```
// Penalize crossing from south to north with first parameter.  
if (isSouth && (ZoneTo.getTTFunctionParam() == North))  
    the_time += params[0];  
  
// Penalize crossing from east to west with second parameter.  
if ((ZoneFrom.getTTFunctionParam() == East) && (ZoneTo.getTTFunctionParam() == West))  
    the_time += params[1];  
}  
  
// Do not return a negative travel time.  
if (the_time < 0.0)  
    return 0.0;  
  
return the_time;  
}  
  
#else  
  
// Generic travel time function...doesn't do anything.  
#include <math.h>  
  
#include <ACT/ACTError.h>  
#include <ACT/TravelTimeFunction.h>  
#include <ACT/Zone.h>  
#include <GBL/TLog.h>  
#include <GBL/Random.h>  
  
// Get the travel time for the given mode using the function  
// supplied in the method. Returns travel time in seconds.  
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time,  
    TZone& ZoneFrom, TZone& ZoneTo, REAL fromX, REAL fromY, REAL toX, REAL toY,  
    TLog& logstream, TRandom& randomstream, vector<double>& params)  
{  
    return 0.0;  
}  
#endif
```

3.4 TravelTimeFunction.C.SN_WE

Travel Time Function for South/North and West/East Bridge Crossings

```
#ifdef CASESTUDY

// Portland travel time function based on river crossings
// and zone to zone travel times from file.
#include <vector>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <ACT/ACTError.h>
#include <ACT/TravelTimeFunction.h>
#include <ACT/Zone.h>
#include <IO/actio.h>
#include <GBL/TLog.h>
#include <GBL/Log.h>
#include <GBL/Random.h>

int region1[] = {999, 970, 982, 1232, 1225, 1226, 1221, 1231, 1234, 997, 998,
    994, 991, 1059, 1058, 995, 1054, 1053, 993, 992, 983, 988, 989, 1051,
    1052, 990, 1050, 984, 1043, 985, 986, 987, 1038, 1039, 1040, 1042,
    1041, 978, 977, 979, 980, 981, 1017, 1018, 1027, 1029, 1028, 1021,
    1022, 1019, 1020, 973, 1005, 1006, 972, 1012, 971, 1007, 1003, 1008,
    1004, 1000, 974, 975, 976, 1049, 1063, 1048, 1047, 1046, 1057, 1037,
    1036, 1044, 1035, 1045, 1031, 1032, 1033, 1030, 1023, 1024, 1025,
    1013, 1014, 1015, 1009, 1010, 1001, 1062, 1149, 1139, 1235, 1233,
    1220, 1117, 1116, 1118, 1115, 1119, 1194, 1197, 1203, 1187, 1188,
    1198, 1204, 1201, 1205, 1196, 1207, 1206, 1195, 1208, 1193, 1209,
    1211, 1210, 996, 1072, 1065, 1060, 1055, 1064, 1034, 1136, 1011,
    1016, 1002, 1185, 1244, 1241, 1243, 1242, 1240, 1236, 1227, 1230,
    1229, 1114, 1228, 1237, 1224, 1110, 1113, 1111, 1222, 1223, 1112,
    1174, 1219, 1108, 1109, 1213, 1214, 1215, 1216, 1217, 1218, 1173,
    1172, 1212, 1171, 1107, 1167, 1102, 1168, 1105, 1170, 1169, 1239,
    1238, 1177, 1176, 1175, 1106, 1103, 1084, 1104, 1096, 1097, 1098,
    1082, 1166, 1101, 1083, 1081, 1100, 1080, 1099, 1092, 1093, 1094,
    1095, 1078, 1079, 1075, 1089, 1076, 1077, 1090, 1069, 1091, 1086,
    1087, 1088, 1073, 1074, 1066, 1070, 1085, 1071, 1061, 1056, 1162,
    1163, 1164, 1165, 1160, 1161, 1159, 1178, 1158, 1157, 1067, 1068,
```

```
1152, 1153, 1154, 1155, 1156, 1147, 1148, 1179, 1150, 1151, 1146,
1144, 1143, 1145, 1140, 1141, 1142, 1183, 1137, 1138, 1133, 1134,
1135, 1184, 1026, 1129, 1130, 1132, 1125, 1131, 1126, 1127, 1128,
1186, 1121, 1199, 1122, 1123, 1124, 1120, 1200, 1191, 1190,
1192, 1182, 1180, 1189, 1181, 1245, 1246};

int region2[] = {270, 271, 272, 273, 268, 274, 269, 275, 276, 206, 277, 236,
267, 207, 167, 279, 278, 208, 205, 166, 129, 168, 204, 280, 223,
281, 169, 237, 234, 235, 259, 258, 225, 224, 282, 103, 130, 171,
24, 209, 165, 210, 170, 164, 262, 203, 131, 257, 233, 212, 226,
211, 213, 286, 260, 172, 227, 232, 104, 163, 202, 261, 256, 238,
201, 102, 197, 162, 132, 266, 198, 264, 265, 239, 263, 215, 105,
173, 255, 231, 128, 240, 101, 200, 199, 214, 194, 174, 133, 195,
161, 283, 284, 251, 252, 253, 285, 250, 254, 192, 196, 228, 249,
241, 193, 242, 216, 79, 175, 106, 230, 127, 191, 229, 136, 160,
108, 80, 190, 100, 218, 217, 107, 134, 243, 301, 137, 99, 159, 81,
82, 135, 244, 110, 189, 188, 176, 109, 177, 187, 299, 138, 126, 111,
245, 98, 219, 39, 139, 186, 112, 185, 184, 246, 140, 178, 141, 158,
157, 125, 83, 155, 97, 96, 183, 302, 300, 142, 303, 179, 156, 124,
144, 113, 145, 143, 84, 154, 220, 85, 248, 182, 123, 87, 86, 180,
146, 95, 122, 247, 153, 114, 287, 94, 298, 181, 91, 121, 152, 147,
304, 305, 88, 115, 306, 120, 148, 297, 118, 93, 116, 221, 90, 151,
92, 117, 89, 222, 316, 149, 119, 308, 310, 317, 309, 295, 150, 339,
315, 340, 296, 933, 318, 314, 319, 311, 288, 313, 932, 320, 307, 338,
402, 403, 312, 336, 321, 337, 341, 294, 322, 335, 332, 344, 342, 289,
394, 334, 333, 343, 345, 348, 346, 349, 350, 347, 365, 366, 293, 362,
363, 364, 291, 367, 290, 373, 368, 371, 372, 370, 292, 369, 374, 396,
397, 392, 391, 395, 399, 384, 393, 398, 383, 67, 361,
930, 26, 27, 25, 19, 23, 20, 22, 17, 18, 21, 8, 7, 28, 29, 5,
30, 6, 9, 4, 3, 962, 10, 1, 2, 31, 11, 12, 34, 13, 32, 16, 33, 15,
14, 36, 44, 35, 38, 43, 42, 37, 47, 48, 46, 46, 948, 40, 51, 41, 50, 947,
52, 946, 945, 53, 57, 56, 49, 58, 54, 55, 59, 61, 64, 60, 62, 63,
944, 65, 66, 70, 71, 69, 72, 73, 75, 68, 74, 77, 76, 78, 325, 326,
327, 324, 323, 328, 329, 331, 330, 353, 355, 354, 352, 351, 356, 360,
357, 359, 375, 358, 377, 378, 376, 379, 380, 381, 382, 385, 400, 390,
931, 520, 386, 389, 387, 388, 401, 45, 1247, 1248, 1249, 1250, 1251, 1252,
1253};

int region3[] = {925, 926, 911, 961, 909, 960, 924, 923, 910, 908, 912, 904,
963, 959, 921, 922, 918, 917, 920, 915, 907, 919, 913, 906, 916, 958,
914, 905, 902, 903, 852, 968, 885, 886, 884, 880, 881, 957, 956, 882,
883, 964, 969, 887, 879, 955, 966, 878, 954, 877, 876, 875, 874, 873,
872, 888, 967, 965, 892, 864, 865, 953, 952, 867, 866, 870, 868, 871,
869, 889, 851, 951, 863, 862, 861, 860, 857, 859, 890, 858, 891, 929,
716, 853, 854, 855, 928, 722, 720, 856, 950, 848, 714, 715, 721, 717,
718, 771, 847, 849, 767, 766, 846, 719, 768, 769, 772, 770, 765, 764,
763, 774, 727, 773, 781, 783, 784, 786, 779, 777, 782, 949, 785, 787,
780, 778, 776, 775, 788, 790, 791, 798, 799, 800, 789, 793, 807, 808,
815, 816, 792, 802, 797, 801, 806, 814, 817, 809, 794, 795, 813, 804,
796, 805, 803, 810, 818, 811, 830, 831, 827, 826, 823, 812, 822, 819,
828, 825, 824, 821, 820, 833, 832, 829, 845, 838, 844, 839, 841, 842,
843, 834, 835, 836, 837, 468, 840, 418, 556, 555, 927, 1292, 678, 633,
679, 677, 634, 631, 702, 703, 637, 680, 676, 635, 675, 681, 701, 700,
```

636, 682, 647, 648, 699, 698, 704, 638, 646, 649, 650, 674, 639, 683,
 673, 651, 697, 645, 671, 672, 684, 695, 696, 644, 652, 705, 643, 708,
 670, 640, 642, 694, 654, 685, 669, 655, 653, 668, 656, 657, 658, 667,
 641, 605, 686, 660, 659, 666, 709, 585, 687, 693, 661, 665, 664, 584,
 663, 688, 692, 706, 662, 689, 707, 691, 690, 542, 583, 550, 901, 900,
 894, 899, 895, 893, 898, 607, 723, 624, 606, 896, 617, 897, 618, 623,
 616, 608, 622, 619, 724, 609, 625, 615, 620, 621, 610, 626, 614, 630,
 725, 627, 611, 612, 613, 628, 629, 747, 746, 726, 748, 762, 761, 599,
 597, 598, 745, 596, 749, 573, 744, 760, 600, 595, 728, 743, 742, 759,
 572, 593, 601, 750, 594, 602, 751, 752, 758, 713, 571, 740, 741, 753,
 591, 592, 729, 603, 754, 757, 588, 590, 730, 574, 755, 739, 570, 850,
 756, 575, 569, 568, 567, 587, 589, 604, 566, 563, 737, 576, 738, 586,
 577, 731, 732, 736, 562, 578, 579, 733, 581, 564, 735, 561, 580, 734,
 558, 559, 560, 711, 565, 712, 710, 582, 632, 1290, 470, 476, 539, 541,
 466, 467, 540, 469, 551, 474, 406, 940, 557, 408, 413, 407, 422, 440,
 442, 436, 410, 411, 412, 465, 471, 941, 443, 939, 423, 943, 942, 409,
 424, 477, 425, 435, 475, 426, 545, 428, 549, 464, 444, 427, 432, 445,
 472, 544, 543, 548, 433, 473, 463, 434, 478, 480, 552, 481, 447, 553,
 429, 479, 430, 446, 461, 431, 486, 448, 485, 458, 459, 449, 488, 487,
 547, 484, 451, 452, 489, 450, 546, 482, 490, 462, 491, 483, 404, 460,
 497, 454, 498, 1291, 453, 554, 457, 496, 503, 938, 502, 495, 494, 538,
 456, 455, 504, 534, 493, 492, 501, 937, 500, 505, 499, 506, 934, 936,
 509, 510, 935, 529, 508, 511, 514, 515, 507, 530, 513, 533, 516, 512,
 531, 517, 526, 527, 518, 523, 519, 536, 537, 524, 522, 528, 521,
 532, 525, 535, 416, 415, 405, 439, 441, 419, 438, 420, 417, 414, 437,
 421, 1254, 1255, 1256, 1257, 1258, 1259, 1260};

```
char ttFtravel_time_file[1024];

/** Travel Times by zone pairs for Portland. ***/
typedef struct times_s {
    REAL times[1293][1293];
} TIMES;

// SOUTH to NORTH and WEST to EAST Penalty
// Travel time function for Portland. param[0] is a constant
// to be added to the travel time for trips that cross bridges
// (regions) south to north (Regions 2 and 3 into Region 1).
// param[1] is a constant to be added for trips that cross bridges
// (Willamette) (Region 2 to Region 3, west to east).
// Uses travel time file to determine zone to zone travel times
// for car, bike, transit, and walk modes.
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time, TZone& ZoneFrom, TZone& ZoneTo, REAL
                                         fromX, REAL fromY, REAL toX, REAL toY, TLog& logstream, TRandom& randomstream, vector<double>& params)
{
    static map<int, short> region;
```

```

static data_loaded = 0;
// Zone to zone travel times by time intervals.
static TIMES auto_tt[5];           // Mode 2
static TIMES transit_tt[5];         // Mode 3
static TIMES walk_tt[1];            // Mode 1
static TIMES bike_tt[1];            // Mode 7
static const int North = 1;
static const int West = 2;
static const int East = 3;
int index = 0;
int params_okay = 0;

// Need two travel time constants in the parameters arrays to
// penalize the bridge crossings.
if (params.size() == 2)
    params_okay = 1;

if (!data_loaded) {
    memset(auto_tt, 0, 5*sizeof(TIMES));
    memset(transit_tt, 0, 5*sizeof(TIMES));
    memset(walk_tt, 0, sizeof(TIMES));
    memset(bike_tt, 0, sizeof(TIMES));
    logstream.print(LOG_ACT) << "TTravelTimeFunction: Loading travel times from file" << done;
    FILE *fp = fopen(ttFtravel_time_file, "r");
    if (!fp) {
        logstream.fatal(SUB_ACT, TERR_ACT_UNREADABLE_FILE)
        << "TTravelTimeFunction: Failed to open zone to zone travel times file "
        << ttFtravel_time_file << done;
    }
    while (moreTravelTimes(fp)) {
        const TTravelTimeEntry *t = getTravelTimeEntryFromFile(fp);
        if (!t)
            continue;
        switch (t->fMode) {
            case 1:
                // Walk - one time interval for walk mode.
                walk_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
                break;
            case 2:
                // Car - 5 time intervals for car mode (1 - 5).
                // Array indexes 0 - 4.
                if (t->fInterval < 6)
                    auto_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
                break;
            case 3:
                // Transit - 5 time intervals for transit (1 - 5).
                // Array indexes 0 - 4.
                if (t->fInterval < 6)
                    transit_tt[t->fInterval - 1].times[t->fZone1][t->fZone2] = t->fValue;
        }
    }
}

```

```

        break;
    case 7:
        // Bicycle - 1 time interval for bicycle.
        bike_tt[0].times[t->fZone1][t->fZone2] = t->fValue;
        break;

    default:
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Unexpected mode "
        << t->fMode << " for zones/interval " << t->fZone1
        << "/" << t->fZone2 << "/" << t->fInterval << " ignored." << done;
        break;
    }
}
data_loaded = 1;
logstream.print(LOG_ACT) << "TTravelTimeFunction: Finished loading travel times from file" << done;
fclose(fp);
}

if (region.empty()) {
    for (int i = 0; i < sizeof(region1) / sizeof(int); ++i)
        region[region1[i]] = 1;
    for (int i = 0; i < sizeof(region2) / sizeof(int); ++i)
        region[region2[i]] = 2;
    for (int i = 0; i < sizeof(region3) / sizeof(int); ++i)
        region[region3[i]] = 3;
}

if (ZoneFrom.getTTFunctionParam() == -1) {
    if (!region.count(ZoneFrom.getNumber())) {
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Zone " << ZoneFrom.getNumber()
        << " missing from region list." << done;
        return 0.0;
    }
    ZoneFrom.setTTFunctionParam(region[ZoneFrom.getNumber()]);
}

if (ZoneTo.getTTFunctionParam() == -1) {
    if (!region.count(ZoneTo.getNumber())) {
        logstream.warning(SUB_ACT) << "TTravelTimeFunction: Zone " << ZoneTo.getNumber()
        << " missing from region list." << done;
        return 0.0;
    }
    ZoneTo.setTTFunctionParam(region[ZoneTo.getNumber()]);
}

REAL the_time = 0.0;

```

```

switch (mode) {
    case 1:
        // Walk
        the_time = walk_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 2:
        // Car
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)
            index = 4;
        the_time = auto_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 3:
        // Transit
        index = 0;
        if (time >= 0.0 && time < 6.5)
            index = 0;
        else if (time >= 6.5 && time < 8.5)
            index = 1;
        else if (time >= 8.5 && time < 16.5)
            index = 2;
        else if (time >= 16.5 && time < 18.5)
            index = 3;
        else if (time >= 18.5)
            index = 4;
        the_time = transit_tt[index].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    case 7:
        // Bike
        the_time = bike_tt[0].times[ZoneFrom.getNumber()][ZoneTo.getNumber()];
        break;
    default:
        break;
}

// Penalize travel time between regions.
if (params_okay) {
    int isSouth = 0;
    if ((ZoneFrom.getTTFUNCTIONParam() == West || ZoneFrom.getTTFUNCTIONParam() == East))
        isSouth = 1;
}

```

```
// Penalize crossing from south to north with first parameter.  
if (isSouth && (ZoneTo.getTTFunctionParam() == North))  
    the_time += params[0];  
  
// Penalize crossing from west to east with second parameter.  
if ((ZoneFrom.getTTFunctionParam() == West) && (ZoneTo.getTTFunctionParam() == East))  
    the_time += params[1];  
}  
  
// Do not return a negative travel time.  
if (the_time < 0.0)  
    return 0.0;  
  
return the_time;  
}  
  
#else  
  
// Generic travel time function...doesn't do anything.  
#include <math.h>  
  
#include <ACT/ACTError.h>  
#include <ACT/TravelTimeFunction.h>  
#include <ACT/Zone.h>  
#include <GBL/TLog.h>  
#include <GBL/Random.h>  
  
// Get the travel time for the given mode using the function  
// supplied in the method. Returns travel time in seconds.  
REAL TTravelTimeFunction::getTravelTime(int mode, REAL time,  
    TZone& ZoneFrom, TZone& ZoneTo, REAL fromX, REAL fromY, REAL toX, REAL toY,  
    TLog& logstream, TRandom& randomstream, vector<double>& params)  
{  
    return 0.0;  
}  
#endif
```